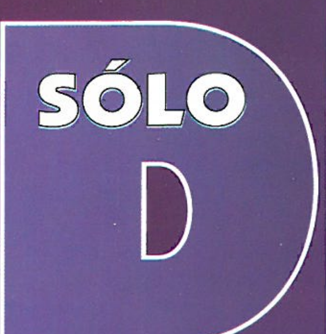


ROGRAMADORES

Revista especializada para usuarios de PC

AÑO 2. Nº 16
1250 PTAS.
ARGENTINA 9'50 $
CHILE 3000 $





Y además:
- Cómo se hace una Intro
- Programación de sonido en Linux
- UNIX: El sistema de ficheros
- ADABAS
- Juegos Conversacionales

Hipertexto: más que un sistema de ayuda

Reportaje: Euskal Party 95

El Lenguaje HTML

El diseño de un lenguaje de programación

TORNEO DE PROGRAMACIÓN DE DEMOS

250.000 Ptas. en efectivo para el ganador

RAY TRACING: LA REFLEXIÓN DE LA LUZ







AHORA
YA
PUEDE
ABRIR
SUS
PROPIAS
WINDOWS

ahora bien, WINDOWSI!
...PERO CON APELLIDOS :

Window Base 2.0

EL SISTEMA DE GESTIÓN DE BASES DE DATOS RELACIONAL BAJO WINDOWS

SEDYCO, S.A. - Distribuidor Mayorista Exclusivo para España de SPI, Inc.
Francos Rodríguez, 64 - 7º B Esc. Izq. • 28039 Madrid • Tel. (91) 311 46 45 • Fax (91) 450 04 24 • E-Mail: sedyco@ibm.net

© SEDYCO, S.A.

DALI (1925) - "Muchacha a la ventana"






DICIEMBRE 1995. Número 16
SÓLO PROGRAMADORES es una publicación de Tower Communications

Editor
Antonio M. Ferrer Abelló
Director Editorial
Mario de Luis García
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer

Director
Mario de Luis García
Redactor Jefe
Carlos Doral Pérez
Coordinador técnico
Eduardo Toribio Pazos
Publicidad
Mª Eugenia González
Colaboradores
Fernando de la Villa, Fernando J. Echevarrieta,, Pedro Antón. Juan M. Martín, Luis Martín, José María Peco, José Carlos Remiro, Enrique de Alarcón, Jorge R. Regidor, Agustín Guillén, Juan Ramón Lehmann, Héctor Martínez, Álvaro Silgado, David Aparicio, Ignacio Cea, Enrique Castañón, Daniel Navarro.
Maquetación
Fernando García Santamaría
Tratamiento de imagen
María Arce Giménez
Servicio Técnico
Oscar Rodriguez
Servicios Informáticos
Digital Dreams Multimedia, S.L.
Ilustraciones
Miguel Alcón
Secretaría de Redacción
Rosa Arroyo
Suscripciones
Erika de la Riva
Redacción, Publicidad y Administración
C/ Marqués de Portugalete, 10
28027 MADRID
Telf.: 741 26 62 / Fax: 320 60 72
Filmación
Filma Dos S.L.
Impresión
GRAFUR, S. A.

La revista SÓLO PROGRAMADORES no tiene por qué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados.
El editor prohibe expresamente la reproducción total o parcial de los contenidos de la revista sin su autorización escrita.

Depósito legal: M-26827-1994
ISSN: 1134-4792

CADA VEZ SE PROGRAMA MÁS ¿O NO?

En las vísperas de comenzar otro año, en SÓLO PROGRAMADORES la redacción hace la *Reflexión Anual.* Y ¿Qué es eso? Os preguntaréis algunos. Pues ni más ni menos que un repaso a la andadura editorial que hemos recorrido en este 1995. Cuando nació SP, allá por septiembre de 1994, todos esperábamos hacer una revista de y para programadores. Nada que tuviera algo que ver con el resto de publicaciones que había en el mercado. Y lo conseguimos. La respuesta favorable de todos vosotros fue unánime. Pero lo mejor, estaba aún por llegar. Ninguno de nosotros podía imaginar que el día a día de la revista lo íban a hacer los lectores con sus cartas, sugerencias, ideas para nuevos artículos, etc. Prácticamente no ha sido necesario pensar, sólo escuchar y hacer caso de vuestras inquietudes y necesidades. Mención aparte se merecen los universitarios, de las carreras más variadas que han elegido SP como complemento a su formación, en una asignatura en la que los límites los pone cada lector. Todos, profesionales, aficionados, estudiantes, se incorporaron a la gran familia de SÓLO PROGRAMADORES. El resultado: cada día crece la afición por la programación y hay más gente desarrollando software y la mejor prueba, es el gran número de empresas españolas desarrolladoras que se han presentado en el pasado SIMO.

Bueno, reflexiones aparte, estamos orgullosos de seguir contando con todos vosotros mes a mes. En este número y a petición de los lectores incluimos un CD ROM de regalo con el GNU C++ para MS DOS. Sin duda alguna, uno de los mejores compiladores de C. Además, en el CD también hay varias librerías de programación que utilizaremos en números próximos para documentar ejemplos de futuros artículos. De momento, dejadlas ahí, aunque los más atrevidos podéis ir experimentando ya con ellas.

Pasamos a otro tema *popular:* las fichas que SÓLO PROGRAMADORES regala cada mes. En el número anterior, por motivos ajenos a nuestra voluntad, no se incluyeron las *codiciadas* fichas y en este, como podréis comprobar, tampoco. A partir del próximo mes continuaremos con las entregas de forma regular, y con nuestra política de siempre de complementos y regalos. Esperamos contentar a todos a la vez, entre libros, fichas y CDs. Así que un poco de paciencia.

Windows 95 sigue dando mucho que hablar, en esta ocasión por sus cuelgues y errores. El Servicio Técnico de Microsoft recibe un gran número de llamadas. Las solución a todos los problemas seguramente llegará en el primer trimestre de 1996 y se llamará Windows 96. Mucha gente opina que OS/2 es un sistema operativo mucho más fiable que Window 95. De hecho, las aplicaciones críticas (medicina, defensa) corren bajo plataformas en las que está instalado OS/2. Sin embargo, IBM no consigue, por más que se lo proponga, llegar al gran público con sus sistemas operativos y es Microsoft quien, gracias a sus extraordinarias campañas de *marketing,* se está llevando *el gato al agua.*

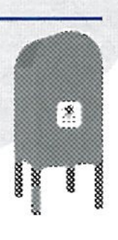
Estas navidades prometen ser un periodo de inversión y ampliación del equipo informático. Según las encuestas, el CD ROM será el periférico por excelencia elegido entre todos los usuarios. También habrá más de uno que aproveche para aumentar la RAM de su ordenador. Yo, personalmente también me he planteado la idea de mejorar mi equipo, cosa que recomiendo a todos de forma encarecida. La mayoría de las veces, el dinero invertido en un equipo rápido se recupera con creces con el ahorro en tiempo de desarrollo. El *streamer* o cualquiera de los dispositivos de *backup* existentes en la actualidad también son una opción interesante.

Nada más por este mes, felicitaros las navidades y agradeceros haber seguido este año (y los que vengan) con nosotros. Un saludo.

MARIO DE LUIS

SUMARIO





EN PORTADA



Tercera entrega del curso de Ray Tracing, este mes se estudia el fenómeno de la reflexión de la luz. Se abre un nuevo abanico de posibilidades visuales con resultados sorprendentes

EUSKAL PARTY 95



Sólo Programadores se desplazo al País Vasco donde se celebró la ya tradicional Euskal Party. Reportaje e información en páginas interiores

CONTENIDO DEL CD-ROM

En el CD-ROM que regalamos este mes, ofrecemos el entorno de desarrollo GNU completo para 32 bits, con el código fuente incluido, para desarrollar aplicaciones profesionales en entorno MS-DOS. Se trata de un compilador de C, otro de C++, un depurador, un generador de proyectos (MAKE), YACC y LEX, un extensor de 32 bits con su propio gestor de memoria extendida y virtual, emulador de 387, librerías de programación de VGA y sonido y documentación de todo ello.

En un artículo pasado de la sección de Linux, hablamos algo sobre el proyecto GNU. Se trata de ofrecer un sistema operativo completo, con fuentes disponibles, y que pueda correr en multitud de plataformas diferentes. Entre otras cosas, cuenta con su propio entorno de desarrollo, en el que se puede reconstruir el sistema entero. Linux es un ejemplo de sistema donde parte de las herramientas se toman del proyecto GNU. Pues bien, lo que distribuimos este mes es todo el sistema de desarrollo, adaptado al sistema operativo MS-DOS, y que se denomina DJGPP. Aunque tendremos algunas limitaciones, como la imposibilidad de utilizar llamadas que impliquen multitarea (primitiva fork), podremos disponer de un potente y completo entorno de desarrollo de aplicaciones 32-bits, con todas las fuentes a nuestra disposición, sin necesidad de instalar ningún sistema operativo diferente del que ya tenemos.

Los requerimientos para nuestro PC son:
386SX o superior, con 512 Kbytes de RAM se recomienda RAM extendida o expandida adicional.
Desde 4 hasta 70 Mbytes de disco duro según instalación.
Soporta SVGA 1024x768 en 256 colores.
Programas de hasta 128 megas, con memoria virtual.
Soporta VCPI y DPMI.

INSTALACIÓN

Entre la multitud de ficheros disponibles para nuestro nuevo entorno, necesitaremos instalar un mínimo para poder comenzar a trabajar.

Para el ejemplo que sigue, vamos a suponer que el usuario tiene un disco duro (C:) y una unidad de cd-rom (D:) donde se encuentran los ficheros de instalación. Los lectores que tengan otra configuración deberán tenerlo en cuenta para adaptar los comandos de instalación a su propio entorno.

En primer lugar, se debe crear un directorio base en nuestro disco duro (en nuestro ejemplo, CC), a partir del cual descomprimiremos los archivos que nos interesen.

C:> MD CC
C:> CD CC
C:> CD D:\DJGPP
C:> D:UNZIP-DJ -o D:DJEOE111.ZIP
C:> D:UNZIP-DJ -o D:DJDEV111.ZIP
C:> D:UNZIP-DJ -o D:GCC257BN.ZIP
C:> D:UNZIP-DJ -o D:GAS211BN.ZIP
C:> D:UNZIP-DJ -o D:BNU22BN.ZIP
C:> COPY D:GO32.EXE \CC\BIN

Si aparece el mensaje "Sobreescribir" habrá que contestarle afirmativamente.

El descompresor UNZIP-DJ viene incluido en el mismo subdirectorio del cd-rom que el propio compilador.

Tras realizar el proceso anterior, se habrán instalado todos los ficheros necesarios para utilizar el compilador, junto con la estructura de directorios correspondiente.

Si deseamos descomprimir otros archivos para instalar material adicional, nos puede servir de ayuda el saber que los ficheros **BN.ZIP* son binarios, los **DC.ZIP* son documentación, y los **S?.ZIP* son de fuentes.

Se debe modificar el *CONFIG.SYS* con las siguientes líneas:
files=15
shell c:\command.com c:\ /e:1000 /p

Los valores 15 y 1000 son los mínimos recomendados. Si se compilan proyectos muy grandes, puede ser necesario aumentar dichos parámetros.

El fichero AUTOEXEC.BAT debe contener los siguientes cambios:
PATH=....;C:\CC\BIN;
call SETDJGPP c:\CC C:/CC

Es decir, que el lector deberá incorporar el subdirectorio donde se encuentran los ejecutables del compilador a la ruta de acceso que ya tenga definida, y luego se deberá invocar un fichero batch para que inicialice el entorno.

Tras rearrancar, el compilador ya debe estar instalado. Hay algunos ficheros de ejemplo para probar que todo está bien:

cd \CC\SAMPLES\HELLO
y teclee TESTIT

Se nos informará de los módulos no instalados pero que pudieran resultar interesantes. Con la instalación anterior, nos faltará, por ejemplo, el compilador C++. El lector que desee incorporar estos programas, deberá descomprimir los archivos correspondientes, en base a la descripción que proporcionamos en el último apartado y con la información que proporciona el fichero README.DJ, en el propio cd-rom.

Pulsando la tecla de retorno, se debe compilar correctamente un pequeño programa de ejemplo. Si se produce algún

error, seguramente nos habremos saltado algún punto de la instalación, y convendrá repasarlo todo.

Si todo ha funcionado correctamente, observaremos que se ha creado una estructura de directorios integrada por:

- *\CC\BIN:* Los ejecutables del entorno. Comprenden, entre otros, el compilador de C (GCC.EXE), el extensor de DOS (GO32.EXE), el linker (LD.EXE), gestor de librerías (AR.EXE), ensamblador (AS.EXE), preprocesador (CPP.EXE) y gestor de proyectos (MAKE.EXE).
- *\CC\DIFFS:* Los cambios introducidos en las anteriores herramientas entre la versión anterior y la que tenemos instalada. Si es la primera vez que vemos este entorno, no tiene gran utilidad.
- *\CC\DOCS:* Documentación de última hora sobre los ejecutables instalados.
- *\CC\DRIVERS:* Los controladores de vídeo cuando utilizamos la librería gráfica.
- *\CC\INCLUDE:* Los ficheros de declaraciones para uso de librerías.
- *\CC\LIB:* Las librerías disponibles.
- *\CC\LIBSRC:* Últimos cambios sobre las librerías con respecto a versiones anteriores del entorno. Parecido al directorio DIFFS.
- *\CC\MANIFEST:* Aunque disponemos de los fuentes y ejecutables del entorno sin coste alguno, en este directorio se encuentran las claúsulas de uso del compilador, amparado por la licencia GNU.
- *\CC\SAMPLES:* Ejemplos que demuestran las posibilidades del compilador.
- *\CC\TMP:* Directorio utilizado por el entorno para ficheros temporales y zona de intercambio cuando se usa memoria virtual.

COMPILANDO UN EJEMPLO

Para comprender mejor el uso del compilador, vamos a seguir un ejemplo paso a paso. Como los ficheros con extensión *.CC son fuentes de C++, y deberemos instalar componentes adicionales, compilaremos el programa del directorio *SAMPLES\HELLO*

Podemos obtener el ejecutable de forma manual, con los siguientes comandos:

gcc -o hello hello.c

Con este comando compilamos el ejecutable "HELLO", sin extensión. Para utlizarlo, deberemos emplear el extensor de 32 bits, de la siguiente forma:

go32 hello

El compilador genera un formato denominado COFF (similar al ELF). Ambos son formatos binarios genéricos para máquinas Intel x86. Si lo deseamos, podemos convertirlo a un EXE, con:

coff2exe hello

Con esto podemos ejecutar nuestro programa (de 32 K) directamente, pero todavía necesitaremos que GO32.EXE se encuentre en la ruta de búsqueda.

CONTENIDOS

Recordando que los ficheros de la forma **SR?.ZIP* son fuentes, los **DC.ZIP* son documentación y los **BN.DOC* los binarios, enumeraremos el contenido de los archivos que incluimos con el CD:

AEASY102 ZIP: Librería para manejo de puertos serie.
AECUR102 ZIP: Librería portable para manejo de texto.
AETSK102 ZIP: Librería de multitarea cooperativa en C++.
AEWIN102 ZIP: Librería de gestión de ventanas de texto.
CBGR?103 ZIP: La librería gráfica GRX.
BCCGRX12 ZIP: Traductor de gráficos Borland a GRX.
GRX103M.ZIP:* Actualización de la librería GRX.
DJLGR111 ZIP: Una librería gráfica sencilla.
QDDVX102 ZIP: Librerías y utilidades para DesqView/X
SBLASTER ZIP: Ejemplos de programación de la SoundBlaster.
WMEMU111 ZIP: Un emulador mejorado de 387.
BNU22.ZIP:* Utilidades varias para el entorno.
BSN122 ZIP:* Un compilador sintáctico para construir compiladores.
FLX238 ZIP:* Un compilador léxico para generar compiladores.
DJDEV111 ZIP: Sistema básico de desarrollo.
DJEOE111 ZIP: Entorno de ejecución de programas
FSDB091A ZIP: Depurador de pantalla completa.
GAS211 ZIP:* El ensamblador
GCC257 ZIP:* El compilador de C.
GPP257.ZIP: Ejecutable y librerias de C++.
MAK369 ZIP:* Make, el generador de proyectos.
OBJC257.ZIP: Entorno del compilador Objetive-C.
DJTST111 ZIP: Programas de ejemplo.
DJEMU111 ZIP: Fuentes del emulador 387.
DJLSR111 ZIP: Fuentes del sistema básico.
DJSRC111 ZIP: Fuentes de utilidades y GO32
DJ111M.ZIP:* Ficheros de actualización y parches.
LGP250 ZIP:* Fuentes de las librerías de C++.
PAT202SR ZIP: Fuentes de PATCH (para aplicar actualizaciones).
MERGE_SR.ZIP: Une varios fichero fuente en uno.
SPLIT_SR ZIP: Separa un fichero fuente en varios.
DJDOC111 ZIP: Documentación básica.
TXI210 ZIP:* Sistema de navegación por la documentación.

GZP124 ZIP:* El compresor/descompresor ZIP de GNU.
UZP501.ZIP:* Un descompresor ZIP (InfoZip).
ZIP191 ZIP:* Un compresor ZIP (InfoZip).

CONTENIDO SHARE

Hemos incluido un gran conjunto de programas shareware agrupados en varios directorios:

UDOS: utilidades para DOS
UWIN: utilidades para WINDOWS
UOS2: utilidades para OS/2
PROG: librerias de programación (ensamblador, C++, Pascal, etc.)

El directorio REVISTAS contiene cuatro revistas electrónicas que edita la empresa CEAO Multimedia de Palma de Mallorca orientadas a los usuarios de Visual Basic, Excel y Access.

En DISC16 encontraréis el contenido del disquete habitual de Sólo Programadores con los fuentes de los artículos.

SÓLO PROGRAMADORES

NOTICIAS

Procesador OverDrive Pentium a 83 Mhz

Intel Corporation ha anunciado recientemente la disponibilidad del nuevo procesador OverDrive Pentium a 83 Mhz, diseñado para actualizar los ordenadores equipados con procesadores IntelDX2 - 66 Mhz e Intel486 - 33 Mhz a la tecnología Pentium.

OverDrive Pentium es un procesador pensado para obtener un mejor rendimiento con los sistemas operativos más exigentes, como Windows 95, Windows NT y OS/2 WARP, así como para las nuevas aplicaciones multimedia, de comunicación y juegos.

Entre sus principales características destacan las siguientes:

- Tecnología del procesador Pentium en 3,3 voltios y 0,6 micras.
- Arquitectura superescalar.
- Predicción de ramificación.
- Coprocesador matemático más rápido.
- Memoria caché integrada de 32 Kbytes.
- Interfaz de bus rediseñado.
- Relación frecuencia bus/corazón de 5:2.

El índice iCOMP del procesador es de 581, lo que supone un incremento del 96% con respecto a una máquina dotada de un procesador IntelDX2 - 66 Mhz (índice iCOMP de 297).

Este OverDrive podrá adaptarse a aquellas máquinas equipadas con un procesador Intel486 con zócalo para 237 pines ("socket 3") o 238 pines ("socket 2"), y ya está disponible al P.V.P. recomendado de 42.200 pesetas (sin IVA).

Para más información:
Intel Corporation Iberia
Tel: (91) 308 25 52
Fax: (91) 310 54 60
World Wide Web: http://www.intel.com/procs/ovrdrive

Chipcom amplia su oferta de soluciones ATM

Chipcom anuncia nuevos productos para su sistema de conmutación Oncore y familia Onsemble. Dichos productos son módulos UNI ATM 155 Mbps para Oncore, un sistema de conmutación de 10 slots Oncore ATM, y el puente Lan-a-ATM Onsemble. Estas soluciones ATM han sido diseñadas para ayudar a los clientes a migrar desde las redes de área local tradicionales a entornos campus conmutados y virtuales de alta velocidad.

El sistema de conmutación Oncore de 10 slots ATM es un backbone de conmutación de alta velocidad para el sistema Oncore de 10 slots, capaz de soportar módulos ATM y futuros módulos de conmutación de paquetes. La porción ATM de este sistema es un backplane basado en switch, y está disponible un backplane opcional instalable para añadir fácilmente capacidades ATM a las plataformas Oncore de 10 slots existentes.

Por otra parte, el puente Onsemble LAN-a-ATM permite una migración de LANs Ethernet y Token Ring a centros de red ATM. El puente dispone de dos puertos ampliables a cuatro y tiene una tarjeta opcional de 100 Mbits por segundo.

Para más información:
Unitronics
Fernando Casado
Tel: (91) 542 52 04

UNIX con servicios de red avanzados y corporativos

La compañías Hewlett-Packard, Novell y The Santa Cruz Operation (SCO) han establecido relaciones empresariales para desarrollar un sistema operativo UNIX con los servicios corporativos de NetWare y UNIX. El objetivo de las tres empresas es ofrecer a sus clientes un camino claro hacia la informática en red de 64 bits en la arquitectura de HP/Intel. A consecuencia de dichas relaciones empresariales se llevará acabo lo siguiente:

- HP dirigirá el sistema operativo UNIX de 64 bits para la arquitectura HP/Intel. Como resultado se combinarán los sistemas operativos SCO de 64 bits y HP-UX basados en Intel.
- Novell ofrecerá sus servicios NetWare y trabajará junto con HP para crear una implementación de alto rendimiento de los NDS (NetWare Directory Services) y de los File/Print Services para HP-UX, además de integrar NDS con DCE.
- SCO, que ha adquirido UnixWare de Novell, consolidará su sistema OpenServer y UnixWare en un sistema UNIX de alto rendimiento basado en Intel que ofrece interfaces en común con HP-UX.

Para más información:
Novell Spain
Daniel Toledano
Tel: (91) 577 49 41

Nuevo motor de estación de trabajo de Btrieve

Btrieve Technologies Inc. (BTI) acaba de anunciar Btrieve for Windows NT/Windows 95. Se trata de una verdadera versión 32 bits de su base de datos navegacional cliente/servidor. El nuevo motor de bases de datos permite la creación y el desarrollo de aplicaciones Btrieve de altas prestaciones para los sistemas operativos mencionados sin modificaciones.

La versión 6.15 de Btrieve para Windows NT/Windows 95 soporta el desarrollo de aplicaciones multitarea de 32 bits, así como la función de registro de Windows NT y Windows 95. Al igual que los demás productos Btrieve 6.15 proporciona, entre otras, las siguientes características: control máximo de programas, roll forward, caching sofisticado de datos y operación sin mantenimiento.

El motor de estación de trabajo para un puesto tiene el precio de 149 dólares, mientras que el de la versión de distribución ilimitada es de 1.995 dólares (licencias de distribución para Windows NT y 95 incluidas). Por su parte, el kit de desarrollo tiene el precio de 595 dólares.

Para más información:
Btrieve Technologies, Europe
35, Cours Michelet
92060 Paris La Défense 10 Cedex - Francia
Tel: 33 (1) 47 73 90 90
Fax: 33 (1) 49 00 01 74

Lanzamiento de INFORMIX-OnLine Extended Parallel Server 8.0

Informix Software Ibérica ha anunciado el lanzamiento en España del gestor de bases de datos INFORMIX-OnLine Extended Parallel Server 8.0 (OnLine XPS). El nuevo gestor es el tercero de una serie de servidores de bases de datos compatibles basados en la arquitectura DSA (Dynamic Scalable Architecture), que se extiende ahora a los clusters de sistemas de Multiproceso Simétrico (SMP) y a los sistemas de Proceso Paralelo Masivo (MPP). De esta forma se aprovechan las ventajas en rendimiento y escalabilidad de las arquitecturas débilmente acopladas (loosely coupled).

Entre las características de OnLine XPS 8.0 destacan las siguientes:
- Funcionalidades para facilitar la gestión de aplicaciones de misión crítica y manejo intensivo de VLDB (Very Large DataBases).
- Escalable y de altas prestaciones.
- Gestión dinámica del sistema.
- Alta disponibilidad y tolerancia a fallos.
- Fragmentación flexible e inteligente.
- Tecnología Parallel Data Query (PDQ) mejorada.

La disponibilidad de OnLine XPS está prevista para el cuarto trimestre del presente año. Las primeras plataformas soportadas serán los servidores MPP de AT&T GIS, IBM RISC System/6.000 SP, ICL Goldrush MegaServer y Siemens Nixdorf RM 1.000, así como los sistemas cluster HP 3.000. A lo largo de 1.996 se ampliará el soporte a otras plataformas.

Para más información:
INFORMIX
Natividad de Mateo
Tel: (91) 372 98 00
Fax: (91) 372 99 12

El P6 cambia de nombre

El procesador conocido hasta ahora por el nombre de "P6" será lanzado a finales de este año con el nombre "Pentium Pro". Intel ha elegido este nombre basándose en el valor de la marca Pentium, expresando que se trata de un procesador que va más allá del original. Pentium Pro es un microprocesador equipado con 5,5 millones de transistores dirigido a las estaciones de trabajo, a los sistemas de sobremesa de gama alta y a los servidores económicos.

Se puede obtener más información sobre el nuevo procesador visitando la página de Intel en el World Wide Web en la dirección URL http://www.intel.com.

Para más información:
Abanico/Hot Line
Liliane Chinyavong
Tel: (91) 594 43 53

Nuevos microcontroladores de Hitachi

Hitachi Europe ha lanzado la serie H8S de microcontroladores de 16 bits de gama alta. Dichos controladores proporcionan 10 Dhrystone MIPS y el conjunto de instrucciones es totalmente compatible con el de la serie de microcontroladores H8/300 y H8/300H de Hitachi.

La serie H8S dispone de una nueva arquitectura mejorada de tipo RISC capaz de realizar las instrucciones básicas en un ciclo de reloj que brinda 2.5 veces las prestaciones de la H8/300H. Proporciona 48 MIPS/Vatio cuando opera a 5 Voltios o 67 MIPS/Vatio cuando lo hace a 2,7 Voltios e incorpora un rápido conversor A/D de 10 bits capaz de realizar conversiones continuas a 1 Mhz. Además, dispone de un interface de memoria mejorado capaz de acceder a memoria DRAM en modo página completa y a la ROM en modo ráfaga.

Hitachi proporcionará un conjunto completo de herramientas de desarrollo para la nueva serie que incluye emulador, debugger y compilador C. Los primeros productos estarán disponibles durante la primera mitad de 1.996.

Para más información:
Hitachi Europe
Pedro Aparicio
Tel: (91) 767 27 82 / 92

NOVEDADES

DMJ

La compañía DMJ especializada en tecnología óptica ha ampliado recientemente su gama de productos mediante un acuerso con Philips Media para la distribución de las herramientas de desarrollo y producción de CD-I (CD Interactivo), vídeo CD y sistemas de compresión de vídeo MPEG.

Con este acuerdo se facilita el acceso a los medios de producción CD-I a todas aquellas empresas que necesiten comunicarse interactivamente. La estación de producción CD-I está basada en el reconocido sistema de autor Media Mogul, un sistema de producción solido y experimentado que ha sido mejorado en la actual versión 2.2 para trabajar sobre PC o Macintosh.



PRESENTACIÓN DE SYSTEM 11.

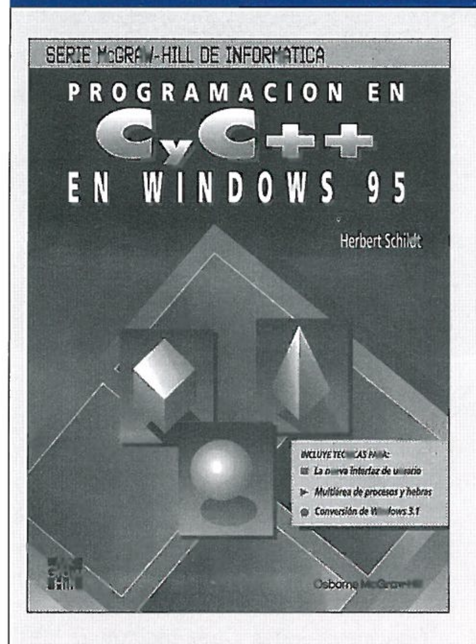
Sybase System 11 agrupa una serie de productos de bases de datos, basados en una única arquitectura dentro de los entornos cliente/servidor.

- **Procesamiento de transacciones en línea (OLTP): SQL Server 11.** Última versión de su sistema de gestión de bases de datos relacionales.
- **Almacenes de datos: Sybase IQ y Sybase MPP.** Amplia sus capacidades ofreciendo una extensión de Sybase SQL Server, basándose en la exclusiva tecnología de indexación Bit-Wise. Sybase MPP, soporta una arquitectura de procesamiento paralelo a gran escala para grandes aplicaciones de almacenes de datos.



- **Distribución masiva de datos: Sybase SQL Anywhere.** Solución orientada a la administración e instalación de bases de datos SQL desde dispositivos móviles, PCs remotos, etc.

LIBROS



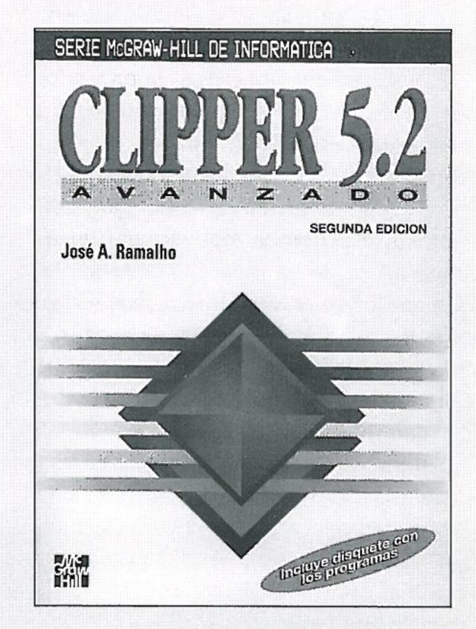
Programación en C y C++ en Windows 95

El conocido autor Herbert Schildt vuelve a la carga con un nuevo libro dirigido a los programadores. En esta ocasión le ha tocado el turno al novedoso Windows 95. Se trata de un libro cuyo objetivo es el de enseñar al programador en entorno Windows las técnicas básicas de programación bajo Windows 95. En la obra se exponen diversas técnicas para la interfaz de usuario, la multitarea de procesos y hebras y la adaptación de aplicaciones de Windows 3.1. Algunos de los temas tratados con mayor detalle son la API Win32, los nuevos controles GUI, el procesamiento de mensajes, los gráficos y los temporizadores. Es un libro claro, estructurado y repleto de código de ejemplo.

Editorial: McGraw-Hill
Autor: Herbert Schildt
438 páginas
Idioma: Castellano
Precio: 3.640 Ptas. (IVA incluido).

Clipper 5.2 avanzado. Segunda edición

Clipper 5.2 avanzado es un libro que trata de forma práctica diversos aspectos de la programación con Clipper, centrándose en las características de la versión 5.2 del lenguaje. El autor trata temas muy variados, como el preprocesador, la herramienta de gestión de proyectos RMAKE, las funciones de bajo nivel y la configuración del entorno de trabajo. Además se examinan los bloques de código, las plantillas y las clases de Clipper 5. Gran parte de las páginas del libro están dedicadas a ejemplos y código fuente. La obra incluye un resumen de las características del lenguaje y un disco con todos los programas de ejemplo.



Editorial: McGraw-Hill
Autor: José A. Ramalho
404 páginas
Idioma: Castellano
Precio: 4.275 Ptas. (IVA incluido).



EL DOLOR DE LA CRISÁLIDA

Cuando entré en contacto con mi primer ordenador, supe que era lo mío. Entrar en su pantalla fue explorar un planeta nuevo y rebosante de posibilidades. Me fascinaba su lenguaje, la manera en que me obligaba a enfrentarme a mis propias contradicciones y poner en orden mis pensamientos. Convertir esa relación en una forma de ganarme la vida no era más que el paso lógico.

Empecé a colaborar con otras personas. Qué distintas son las personas de los ordenadores. A los ordenadores sólo se les puede entrar de frente; por el contrario, la mayoría de las personas son inaccesibles excepto por tortuosos caminos laterales. Con mi pecé sólo existe el lenguaje de la sinceridad. Pero pronto tuve que descubrir que esa sinceridad que, en la soledad de mi habitación, era la única forma de comunicación que conocía, podía ser mortal cuando se trataba de relacionarme con mis nuevos compañeros de carne y hueso. Los ordenadores no perdonan la ambigüedad; la gente, en cambio, no soporta su ausencia.

Fueron poniéndome a las órdenes de distintos superiores. Cada uno de ellos tenía su sello especial. De uno me molestaba su impulsividad y su falta de respeto hacia los sentimientos ajenos. De otro me enternecía su voluntarismo, sus ganas de demostrarnos a los demás y a sí mismo que era capaz de llevar un proyecto adelante. De otra me desconcertaba que me hiciera trabajar durante días para explorar un camino nuevo, y luego me obligara a tirarlo todo y abrir brecha por otro lado. Cada uno reaccionaba de distinta forma ante las dificultades o tenía su propio estilo de intentar motivarme. Nunca vi a dos iguales.

Aun así, todos tenían algo en común: parecían haberse olvidado de programar, de la misma forma que los adultos se olvidan de haber sido niños algún día. Los desafíos técnicos, la belleza de un algoritmo elegantemente simple para un problema aparentemente complicado, o mi fascinación al encontrar la intrincada causa de un error, no les provocaban más que impaciencia. No querían oír hablar de desbordamientos de pila o goteos de memoria. Sólo querían ver el trabajo hecho, rápido y bien. Estaban demasiado ocupados con los problemas políticos, con dificultades humanas de colaboración con los socios del proyecto o con la cuidadosa organización de demostraciones para los peces gordos, en las que sólo se mostraba la parte más estúpida de los programas y se daban respuestas técnicamente imbéciles a preguntas técnicamente pueriles.

Durante el roce con estas personas, yo iba pensando que, de estar yo en su lugar, habría dedicado más tiempo a una tranquila planificación antes de zambullirme a codificar, o que me habría ocupado de construir una infraestructura de programación para que luego no se hubieran duplicado las tareas; pero ésos eran pensamientos en segundo plano, porque yo estaba muy satisfecho de ser un simple programador, y jamás me hubiera imaginado paseándome entre las mesas gesticulando y diciendo "¡Haz esto! ¡Haz aquello!". Eso estaba reservado para ellos, esa especie cromosómicamente diferente, a quienes el nudo de la corbata había reducido el riego sanguíneo del cerebro hasta obsesionarles con presupuestos y plazos y volverles incapaces de distinguir un módulo de C de un poema en noruego.

Han pasado algunos años y me he convertido en una especie de desratizador de lujo. Me gustaría pensar que soy el gurú al que el resto del equipo acude cuando tiene una incertidumbre técnica que puede poner el proyecto en peligro, pero mucho me temo que soy el sabelotodo impertinente al que alguien pregunta cuando le da pereza buscar en el manual. Alguien que me aprecia, y que tiene mucho que ver con que yo esté ahora desahogándome en este espacio, me dijo dos frases que me abrieron los ojos. Una era: "Si te conviertes en aquél que sabe por qué fallan los programas, estás perdido". La otra: "Sobran programadores buenos y baratos. Lo que falta es gente que diga: Yo me hago responsable".

Es la hora de abandonar la tierra de Nunca Jamás. Hay que quitarse el traje de Peter Pan y, mal que nos pese, apretarnos el nudo de la corbata. Hoy hubo una tarea nueva que hacer, y, venciendo el impulso de hacerla yo mismo, tomé a uno de las programadores nuevos, un chaval de éstos que se nota que la ropa se la elige su madre, y delegué en él. Hay que romper el capullo, consumar la metamorfosis o recorrer el via crucis de la decadencia. Sólo espero que, al menos, no se me olvide cómo se programa.

EL LENGUAJE HTML (I)

Fernando J. Echevarrieta

Si bien la principal causa del crecimiento del WWW es su sencillez de uso, la segunda es, quizá, la sencillez de su generación. Según las estadísticas de internet, se tarda alrededor de dos horas en aprender el lenguaje HTML de hipertexto y con él, la posibilidad de tejer una pequeña contribución a la gran tela de araña.

En el presente y el próximo artículo se tratará en profundidad el lenguaje HTML de hipertexto. Con ello, el lector será capaz de realizar sus propias páginas de Web sacándole todo el jugo al lenguaje. En caso de que disponga de un acceso a internet, podrá publicar en la red su trabajo. En caso contrario, mientras lo consigue, le servirá para organizar la información que desee en su ordenador mediante un sistema local hipermedia. Aquellos lectores que se encuentren dentro del primer grupo, dispondrán de dos posibilidades para dar a conocer sus páginas: si su acceso es directo y en su red hay un servidor de www, suele bastar con nombrar la raíz de sus documentos como *index.html* y dejarlos dentro de un directorio concreto (bajo el suyo), generalmente *public_html*. Como esto depende de la configuración del servidor pregunte a su *webmaster*. En caso de que no disponga de servidor de Web, espere un próximo artículo en el que se explicará como instalar uno o deje un mensaje en el servidor del autor: *http://highland.dit.upm.es:8000* que con mucho gusto le colocará su página personal. Si accede a internet a través de un proveedor, diríjase a él.

EVOLUCIÓN DEL LENGUAJE

El lenguaje *HTML* nace en 1991 de manos de Tim Berners-Lee del CERN como un sistema hipertexto con el único objetivo de servir como medio de transmisión de información entre físicos de alta energía como parte de la iniciativa WWW. En 1993 Dan Connelly escribe el primer DTD (Document Type Definition) de SGML describiendo el lenguaje. En 1994 el sistema había tenido tal aceptación que la especificación se había quedado ya obsoleta. Por aquel entonces WWW y Mosaic eran casi sinónimos debido a que el browser Mosaic del NCSA (National Center for Supercomputing Applications) era el más extendido debido a las mejoras que incorporaba. Es entonces cuando nace el *HTML 2.0* en un draft realizado también por Dan Connelly (para mayor información sobre drafts y SGML acuda al número anterior de Sólo Programadores). El crecimiento exponencial que comienza a sufrir el sistema lleva a organizar la First International WWW Conference en Mayo de 1994. El principal avance de la versión 2.0 de HTML es la incorporación de los llamados *forms*, formularios que permiten que el usuario cliente envíe información al servidor y ésta sea recogida y procesada allí. Precisamente con este fin, NCSA presenta la especificación del CGI, Common Gateway Interface, versión 1.0 que define un interfaz entre programas ejecutables y el sistema WWW. Con la incorporación de los forms, aparecen por primera vez campos donde el usuario puede escribir, menús *pull-down* y los denominados *radio-buttons* (pulsadores) integrados en páginas WWW.

Desde entonces, el lenguaje ha seguido creciendo como algo dinámico, como una lengua humana, algo vivo, siendo modificado sobre todo por las personas que lo utilizan. Así, una evolución en el lenguaje suele surgir de una propuesta que es adoptada por algunos clientes (browsers). Con el uso se ve si es eficiente y es adoptada y si es así, finalmente se incorpora al estándar. De este modo, a finales de 1993 se comienza a hablar de *HTML+* propuesto por

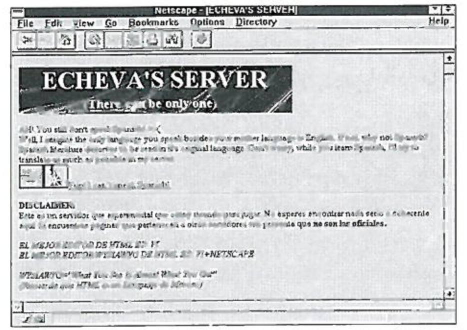


Figura 1. Ejemplo de uso de la etiqueta <IMG>.

Dave Raggett, de H&P Labs. en Bristol que evoluciona a un nuevo draft de Marzo de 1994 para la versión *HTML 3.0* incorporando nuevas posibilidades como la realización de tablas complejas, control de proceso de formatos e incorporación de expresiones matemáticas.

El testigo pasa del browser Mosaic al Netscape, que incorpora nuevas mejoras. Aunque el equipo de Netscape anuncia desde el principio que su browser trata HTML 3.0, lo cierto es que no se adapta al estándar. Por el momento, el único browser de HTML 3.0 es experimental y recibe el nombre de *Arena*. El lenguaje de Netscape, el más utilizado en la actualidad, incorpora etiquetas no definidas en HTML 3.0, y tiene algunas diferencias con algunas de las definidas, por ejemplo en la realización de tablas. Por otra parte, hasta la versión 2.0, recién aparecida, no permitía el empleo de expresiones matemáticas (a la hora de escribir el artículo, el autor aún no ha analizado la versión 2.0). Y como gran idea, propone la incorporación de un tipo MIME experimental que permite la actualización dinámica de documentos, del que se hablará en el capítulo dedicado a la programación de CGI. Por ello, en "los ambientes" se ha comenzado a denominar este lenguaje de Netscape como *NHTML 1.1* para diferenciarlo de la verdadera propuesta de HTML 3.0.

INTRODUCCIÓN AL HTML

Si ha escito alguna vez un texto ASCII, ya ha escrito hace tiempo su primer código HTML. Y es que el HTML es un lenguaje basado en etiquetas que se insertan en lugares estratégicos en un texto ASCII. Una etiqueta HTML viene definida por una palabra o conjunto de palabras entre los signos < y >. Por ejemplo: *<etiqueta>*. La ventaja del lenguaje es que una etiqueta no reconocida es inmediatamente ignorada, por lo que distintos browsers pueden incorporar nuevas etiquetas que sólo ellos entenderán sin que el documento deje de ser compatible con el resto. En general, las etiquetas en HTML aparecen por pares, una de comienzo y otra de terminación, en la que ésta última comienza con un signo / o slash:

<etiqueta>texto</etiqueta>

el olvido de este slash puede producir efectos imprevisibles al no cancelar el efecto de la etiqueta de apertura. Muchas de estas etiquetas podrán también llevar parámetros asociados:

<etiqueta parametro=valor>texto<etiqueta>

Otras dos características a tener en cuenta del lenguaje es que no diferencia las mayúsculas de las minúsculas, por lo que equivale escribir *<etiqueta>* y *<ETIQUETA>*; y que no tiene en cuenta los retornos de carro. Así, las líneas se pueden partir en cualquier momento y aunque se escriba más de un espacio se conviertirán en uno solo. Del mismo modo, al realizar el cliente el procesado del texto, no tendrá en cuenta los retornos de carro, lineas en blanco ni el sangrado que se introduzca en el código HTML. Por ello, se definen las etiquetas *
* (nueva línea), *<p>* (nuevo párrafo) y *<hr>* (línea horizontal separadora).

Como ya se explicó el mes pasado, HTML, como DTD de SGML, se fundamenta principalmente en estilos lógicos. Así, si se etiqueta un título como una *cabecera de nivel 1* (nivel lógico) no llega a especificar como se debe presentar una cabecera de nivel 1 (nivel fisico) por lo que cada cliente o browser la presentará de forma distinta. Diferentes programas con diferentes configuraciones producirán distintas presentaciones de un mismo documento por lo que, en realidad es imposible la existencia de editores *WYSIWYG* (What You See Is What You Get). En general, y aunque el estándar proporcione etiquetas para definir estilos fisicos, es más conveniente utilizar los lógicos. La ventaja es obvia: un texto marcado lógicamente como *<STRONG>* (lógico), puede presentarse, por ejemplo, en negrita (físico) equivalente a *<B>* o en rojo, si así decide configurar su browser el usuario. Sin embargo, si se indica desde el principio *<B>* (físico), quedará determinado su aspecto. Por eso, ¡Casi nadie usa estilos lógicos! Además de que suele ser "más cansado", al diseñador de páginas le suele gustar tener bien controlado el resultado final.

Para poner un ejemplo al lector impaciente, baste escribir en un fichero:

<H1>Esto es una cabecera de nivel 1</H1>
Esta palabra está en <B>negrita</B>
Y ésta está en <I>cursiva</I>

que producirá como resultado algo parecido a:

Esto es una cabecera de nivel 1
Esta palabra está en **negrita**
Y esta ésta está en *cursiva*

Seleccione la opción "open local" u "open file" de su browser de WWW y observe el resultado.

ESTRUCTURA DE UN DOCUMENTO HTML

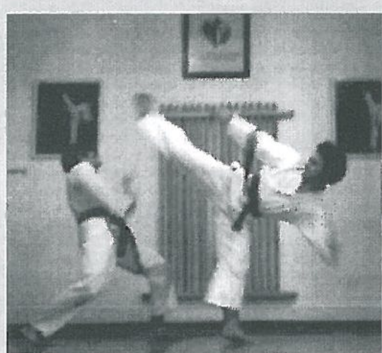
Un documento HTML consta idealmente de dos partes (opcionales): la cabeza, comprendida entre las etiquetas *<HEAD>* y *</HEAD>* y el cuerpo, entre *<BODY>* y *</BODY>*. El objeto de esto es realizar las cosas dentro de un orden, aunque no se verá reflejado en la presentación del documento. Y, puestos a hacer las cosas bien, no estará de más encerrar todo el documento entre etiquetas *<HTML></HTML>*.

En la cabecera suele aparecer siempre una etiqueta *<title>...</title>* con el título del documento. Obsérvese que este título no aparece nunca en el propio documento sino, según el browser,


Para ver mi página personal pulsa aquí o sobre mi imagen en la foto de la trompeta.

Figura 2. Ejemplo de uso de una imagen como enlace hipertexto.

CARACTERÍSTICAS TÉCNICAS



Debido a la influencia de los maestros ITOSHU e IGAONNA, tiene dos formas de trabajo:

- **SHURI**: rápido
- **NAHA**: fuerte.

Posee Katas de *SHURI-TE*, *TOMARI-TE*, *NAHA-TE*. El uso de las posiciones es natural. Ni muy bajas ni muy altas. En las defensas suelen usarse posiciones más bajas que en los ataques. Estos movimientos son reflejados en los KATAS.

Las posiciones más características son: *SANCHIN-DACHI*, *NEIKO-DACHI*, *SHIKO-DACHI*, *ZENKUSHU-DACHI*, *KO-KUTSU-DACHI*, *NEKO-ASHI-DACHI*

Las técnicas de mano abierta, sobre todo en defensa, se usan mucho, y su recorrido es corto. Las técnicas de puño se usan en ataques y contraataques, siendo de recorrido corto y muy rápido.

Las distintas posiciones se utilizan en todas las direcciones coordinando el cuerpo y la cadera con la ejecución de la técnica. Se trabaja mucho en parejas, técnicas de aplicaciones de *KUMITE* controlando al adversario en todo el proceso de la técnica hasta su culminación con un contraataque y con el control total del mismo.



Posee técnicas de defensa clásicas y además técnicas de:

- **GUAKU-WAZA**: Control del adversario.
- **NAGE-WAZA**: Técnicas de proyección y barridos.
- **SHIME-WAZA**: Técnicas de estrangulación.

El trabajo de piernas se realiza *CHUDAN* y *GEDAN* de forma principal pero en los entrenamientos se trabajan también *YODAN* incluso de salto.

Los ataques de avance generalmente son rectos. Las defensas suelen tener una dalida de 45 grados.

Actualizado por: Fernando J. Echevarrieta Fernández. 

Figura 3. Ejemplo de uso de distintas etiquetas básicas.

en el marco de la ventana o en un campo opcional del mismo.

Pero también es posible encontrar otras etiquetas, mucho más infrecuentes, como:

<base href="URL">: Especifica el path del documento. Es útil si las referencias que se tienen de él son relativas.

<link rev="RELACION" rel="RELACION" href="URL">: donde *rev* (reverse) especifica una relación entre el URL y el documento y *rel* una relación entre el documento y el URL. Un ejemplo de esto se puede encontrar en la guía de Michael Grove en cuya cabecera muestra:

<link rev="made" href="mailto:docuhelp@ukanaix.cc.ukans.edu"> para indicar que el creador del documento esta identificado por la URL dada. En cualquier caso, estas etiquetas contienen metatinformación que no afectará al documento, por lo que son desconocidas y ,por lo tanto, no usadas, por la mayoría de los creadores de páginas HTML. Podrá encontrar información sobre URLis en el número anterior de la revista.

Será el cuerpo del documento la parte realmente interesante y donde se podrán incluir todas las etiquetas que aparecen en las tablas salvo las propias de cabecera.

ALGUNOS ESTILOS LÓGICOS

Los estilos lógicos más utilizados son el *énfasis*, que se suele traducir en cursiva; el *código de ordenador*, los *mensajes de ordenador*, *entradas de teclado* y *variables de metasintaxis*, que se suelen traducir en una fuente de caracteres de tamaño fijo. Estos estilos suelen aparecer en manuales hipertexto en frases como las de los siguientes ejemplos:

Teclear: Enter <kbd>passwd</kbd>
rm <var>fichero<var> borra el fichero

De todos modos, son un conjunto de facilidades del estándar que casi nadie usa. Para dar cualquier tipo de formato a sus documentos emplee las etiquetas que figuran en las tablas adjuntas, de las que se explicarán las más relevantes.

IMÁGENES

Una vez se ha dado formato al texto, lo que realmente hará vistosos los documentos será la inclusión de imágenes. Para ello, se define la etiqueta:

<img src="URL" [alt=][align=][ismap]>

Obsérvese que *<IMG>* es una de las pocas etiquetas que no requiere pareja de terminación.

URL localiza la imagen a mostrar.

ALT muestra un texto alternativo en caso de que el browser que se emplee no disponga de capacidades gráficas (por ejemplo, el lynx).

ALIGN indicará la forma de alineamiento que debe tener el texto con la imagen. El estándar inicial define tres tipos: *top*, *middle* y *bottom*, aunque Netscape mejoró notablemente la alineación de las imágenes mediante nuevas opciones de las que las más utilizadas son *left* y *right*. Para conocer todas las extensiones de Netscape diríjase a: *http://home.mcom.com/assist/net_sites/html_extensions.html*

Como ejemplo, puede ver en la figura 1 el resultado de:

<img src="/icons/eche16.gif"

TABLA: CABECERAS

...	Cabecera de nivel 1
...	Cabecera de nivel 2
...	Cabecera de nivel 3
...	Cabecera de nivel 4
...	Cabecera de nivel 5
...	Cabecera de nivel 6

FORMATO GENERAL

<title>...</title>	Título del documento
<p>	Nuevo Párrafo
 	Nueva línea
<hr>	Regla horizontal
<pre>...</pre>	Texto preformateado
<listing>...</listing>	Listado
<blockquote>...</blockquote>	Cita

ESTILOS LÓGICOS

...	Énfasis
...	Refuerzo
<code>...</code>	Código de ordenador
<samp>...</samp>	Mensajes de ordenador
<kbd>...</kbd>	Entrada por teclado
<var>...</var>	Variable metasintáctica
<dfn>...</dfn>	Definición
<cite>...</cite>	Título (libro, film...)

ALT="Echevaís server: There can be only one">

TEJIENDO LA TELA

Al llegar a este punto ya se pueden obtener unos "preciosos" documentos, pero de momento, de hipertexto e hipermedia, nada. Ha llegado el momento de comenzar a colocar los puntos de anclaje en documento de los que colgarán las hebras de los hiperenlaces. Para ello se dispone de la etiqueta *<A>* de Ancla, que puede tomar las formas:

<A NAME="Nombre">...</A>: Identifica al texto como un destino llamado *Nombre*

<A HREF="URL">...</A>: Convierte al texto en un enlace al *URL*

<A HREF="URL#Nombre">...</A>: Convierte al texto en un enlace al destino *Nombre* en el URL

<A HREF="#Nombre">...</A>: Convierte al texto en un enlace al destino *Nombre* dentro del mismo documento.

Lo interesante de esta etiqueta es que todo lo que encierra se convierte en un enlace, de tal forma que si encierra una imagen, al pulsar sobre la misma se producirá el salto. Eso se ha hecho en el presente código, en el que aparece un botón con la cara del autor que remite a su página al ser pulsado y se puede observar en la figura 2:

<A HREF="/echeva/echeva.html"><img src=/icons/echeicos.gif align=middle></A>

< a href="/echeva/echeva.html">aquí o sobre mi imagen en la foto de la trompeta.

En general, se suelen utilizar URLís parciales por grupos de información, como aparecen en los ejemplos del artículo, es decir, indicando la forma de localizar el nuevo item (documento, imagen...) a partir del actual. Esto hace mas fácil su traslado. Los URLís absolutos se suelen emplear para referirse a otro tema lógico (o cuando no queda más remedio, por estar en otra máquina, por ejemplo).

Una de las aplicaciones más inmediatas del estándar CGI y de la que primero se hablará en el artículo correspondiente, es la realización de imágenes sensibles que servirán de enlace a uno u otro lugar según la zona de la imagen sobre la que pulse.

ENTIDADES

Las entidades identifican caracteres cuya representación puede variar de una máquina a otra o son caracteres especiales de HTML. La definicion de una entidad comienza por el símbolo ampersand (&) y termina con un punto y coma (;). Ambos símbolos rodean el signo de la entidad. En el caso del castellano, los caracteres problemáticos son la ñ (*ñ*), las diéresis, ü (*&uuml*), y los acentos, que siempre son agudos (*&aacute*; *&eacute*; ...). Otras entidades son el propio ampersand & (*&*), menor que < (*&lt*); mayor que > (*&gt*), y doble comilla " (*&quote;*). Sin embargo, existen muchas más, sobre todo en HTML 3.0 que incorpora el alfabeto griego para la expresión de matemáticas. Es importante destacar que las entidades sí distinguen entre mayúsculas y minúsculas, por ejemplo, entre *Á* y *á*.

Por tanto, el primer ejemplo que se puso en el artículo, para quedar correctamente definido en todas las plataformas debería escribirse como:

<H1>Esto es una cabecera de nivel 1</H1>

Esta palabra está en <B>negrita</B>

Y ésta está en <I>cursiva</I>

ENTRELAZADO DE ETIQUETAS

En un código HTML es posible entrelazar etiquetas, aunque el resultado obtenido será distinto según el browser empleado. Así pues será posible escribir:

<I>Esto es una <B>prueba</I>. Adios </B>

Aunque algunos browsers lo pueden aceptar, la relación siguiente está prohibida:

<A><H>...</H></A>

siendo la forma correcta:

<H><A>...</A></H>

No se deben poner cabeceras en las listas, aunque si se pueden poner etiquetas de formato.

En general, las etiquetas no son aditivas. Aunque:

<B><I></I></B>

se resolvería correctamente en algunos browsers, en otros únicamente se mostrará el texto en cursiva.

CONCLUSIÓN

En el próximo artículo se profundizará en la realización de documentos más elaborados explicando a fondo la realización de forms, y presentando la forma de realizar imágenes-mapa sensibles, actualización dinámica de documentos, realización de tablas, control de fondos y tipos de letra y algunas extensiones al lenguaje. De momento, vaya practicando con lo expuesto y, como ejemplo, en el disquete se ha incluído el código HTML fuente de la figura 3.

ESTILOS FÍSICOS

...	Negrita
<i>...</i>	Itálica
<u>...</u>	Subrayado
<tt>...</tt>	Máquina de escribir

HIPERENLACES Y ANCLAS

...	Ancla en documento
...	Enlace a ancla
...	Enlace a URL

GRÁFICOS

<img src="URL" [alt=][align=][ismap]>	Imagen

LISTAS Y GLOSARIOS

Una lista se presenta en HTML como:
<etiqueta>
* Item 1*
* Item 2*
...
</etiqueta>
donde <etiqueta></etiqueta> puede ser *
*, *
* (numera los items), *<menu></menu>* o *<dir></dir>*. En un glosario cada item consta de un término definido y una definición por lo que se representa como:
<dl>
<dt> Nombre item 1
<dd> Definición item 1
<dt> Nombre item 2
<dd> Definición item 2
...
</dl>

LISTAS

...	Lista no numerada
...	Lista ordenada
<menu>...</menu>	Menú
<dir>...</dir>	Directorio
	Item en una lista
<dl [compact]>...</dl>	Lista de definiciones
<dt>	Termino en la definición
<dd>	Definicion en la definición

VARIOS

<ló texto ó>	Comentario
<link href="URL" [rev=] [rel=]>	Asociación de metainformación
<address>...</address>	Dirección correo electrónico
<isindex>	Documento es índice
<base>	Path del fichero actual

ORGANIZACIÓN DE ARCHIVOS

José C. Remiro

Gracias a los archivos almacenados en la memoria secundaria es posible el proceso de grandes cantidades de datos. Su organización, así como el conjunto de operaciones básicas que se pueden aplicar sobre éstos es el tema del presente artículo.

La memoria central ha sido hasta ahora el dispositivo donde se almacenaban los datos contenidos en las estructuras que se han presentado en artículos anteriores. La memoria central presenta dos inconvenientes a la hora de almacenar información: Se trata de un recurso escaso del ordenador, por lo que normalmente no se dispone en cantidades suficientes como para almacenar grandes cantidades de datos, es una memoria volátil, es decir, necesita ser alimentada continuamente para que pueda mantener los datos.

Estos dos inconvenientes hacen que la memoria central sea inadecuada para albergar grandes conjuntos de datos y que éstos estén disponibles de forma permanente.

Los dispositivos de almacenamiento externo o memoria secundaria, aunque de acceso más lento que la memoria central, son capaces de albergar permanentemente grandes cantidades de datos, sin embargo, el procesamiento de los datos contenidos en estos dispositivos seguirá realizándose en la memoria central, pero en unidades más pequeñas.

Se puede definir un archivo como un conjunto de datos con las mismas características, que están relacionados y almacenados de manera conjunta en un dispositivo de almacenamiento secundario. Se podrá hacer referencia a un archivo a través de descriptores, éstos tendrán diferentes formas, dependiendo del sistema operativo sobre el que se trabaje.

Los dispositivos de almacenamiento secundario presentan diferentes tipos de acceso a los datos que albergan dependiendo de su naturaleza mecánica. Así, no hay otro modo de acceder a la información presente en una cinta magnética más que en modo secuencial. Esto implica que para acceder a una determinada información presente en la cinta hay que pasar, obligatoriamente, por todos los datos almacenados previamente. Sin embargo, el tipo de acceso que puede realizarse sobre un disco magnético es directo, esto es, para acceder a una determinada información no es necesario haber recuperado la información precedente.

Los diferentes accesos sobre los dispositivos de almacenamiento secundario afectan a los modos en que será posible recuperar la información de los archivos. **Si un archivo está almacenado en cinta, el acceso a éste será obligatoriamente secuencial, por el contrario, si el archivo** se encuentra almacenado en disco se podrá recuperar secuencialmente o de forma directa, en este último caso, indicando de alguna forma el lugar donde se encuentran los datos a recuperar (por ejemplo, en un disco magnético a través del número de pista y del número de sector).

Por cuestiones metodológicas se van a clasificar los archivos en tres clases básicas, éstas están representadas en la mayoría de los lenguajes de programación procedurales de propósito general:

- Archivos de texto.
- Archivos no tipificados.
- Archivos tipificados.

Archivos de texto

Los archivos de texto contienen caracteres y generalmente dos delimitadores, fin de línea y fin de archivo. De forma lógica, se pude pensar que estos archivos están estructurados en líneas, cada línea delimitada por un carácter fin de línea.

Se va a aumentar el seudocódigo hasta ahora utilizado para poder tratar los archivos. Para poder declarar un

identificador de fichero de texto se indicará mediante un identificador, seguido de dos puntos y la palabra reservada **Archivo_Texto**. Para poder realizar la asociación de un identificador de archivo a un archivo determinado se utilizará el procedimiento **Asociar (Ident_Arch, "Descr_Arch")**, donde **Ident_Arch** será un identificador de archivo y **Descr_Arch** será sustituido por un descriptor de archivo.

Por ejemplo, si se está trabajando en MS-DOS y disponemos de un archivo de texto con nombre *c:\arch.txt*, para poder utilizarlo dentro de un programa, se debería definir una variable de tipo **Archivo_Texto** (por ejemplo, Arch: **Archivo_Texto**) y antes de realizar cualquier operación se deberá asignar al archivo la variable anteriormente definida (por ejemplo, Asociar (Arch, "*C:\arch.txt*")).

Cualquier referencia posterior a la variable de tipo **Archivo_Texto**, será en realidad una referencia al archivo al que se le hubiese asociado en último lugar. Posteriormente a la utilización del procedimiento **Asociar** no se hará ninguna **referencia al nombre del archivo, se utilizará el identificador de archivo asociado.**

Antes de realizar cualquier operación de lectura o escritura en el archivo, se indicará mediante el procedimiento **Abrir (Modo_Apertura, Ident_Arch)** que operaciones están disponibles sobre el archivo. Este procedimiento toma un valor de los siguientes para el parámetro **Modo_Apertura**:

- **Entrada.**- Solamente se permite la operación de lectura sobre el archivo. Se indicará sustituyendo **Modo_Apertura** por E.
- **Salida.**- Si el archivo existe, éste se borra, si no existe el archivo se crea pero vacío. Solamente se permite la instrucción de escritura sobre el archivo. Se indicará sustituyendo **Modo_Apertura** por **S**.
- **Adición.**- Se permite la escritura de nueva información pero solamente al final del archivo, siendo necesaria la existencia de éste. Se indicará sustituyendo **Modo_Apertura** por **A**.

Físicamente, mediante el procedimiento **Abrir**, se asocia a un archivo una zona intermedia de memoria, donde se realizarán las transacciones entre la CPU y la memoria auxiliar, cualquier utilización del archivo pasa por trasladarlo en partes suficientemente pequeñas (normalmente en unidades de 512 bytes), realizando en la memoria central las modificaciones pertinentes y cuando se den las condiciones necesarias, se traslada dicha información a memoria auxiliar. Además, el sistema operativo cuando se realiza una apertura de un archivo sitúa apuntadores que indican la posición que se verá afectada al hacer las operaciones pertinentes.

Otro procedimiento común a todos los archivos será **Cerrar (Ident_Arch)**, este procedimiento libera la memoria intermedia asignada al archivo asociado al identificador representado por el parámetro **Ident_Arch**. Este procedimiento se utilizará al finalizar el tratamiento del archivo, si no se utiliza puede ocurrir que el archivo se corrompa.

Los procedimientos **Asociar**, **Abrir** y **Cerrar** son aplicables a todos los tipos de archivo, siendo solamente necesario variar el tipo de parámetros que se pasan a los procedimientos dependiendo del archivo a utilizar.

Las operaciones básicas posibles **sobre un archivo de texto son dos: leer un carácter y escribir un carácter.** Para indicar la lectura en un archivo de texto mediante seudocódigo se utilizará el procedimiento **Leer(Ident_Arch, Ident_Car)**, donde **Ident_Arch** es un identificador de archivo de texto e **Ident_Car** es un identificador de carácter, este procedimiento asigna el carácter al que señala el puntero del archivo y lo almacena en el identificador de carácter, moviendo el puntero al siguiente carácter del archivo.

La escritura de un carácter en un archivo se indicará en seudocódigo mediante el procedimiento **Escribir (Ident_Arch, Ident_Car)**, el efecto sobre el archivo es la escritura del carácter en la posición señalada por el puntero asociado al archivo y el traslado del puntero a la siguiente posición.

Por último, se dispone en el seudocódigo de una función que devuelve un valor de tipo lógico, **EOF(Ident_Arch)**, que devuelve el valor **verdadero** cuando el puntero asociado al archivo señala el final y el valor **falso** en cualquier otro caso. El cuadro 1 muestra el seudocódigo de un programa ejemplo que se encarga de concatenar el archivo PRIMERO con el archivo SEGUNDO, dejando el resultado en el archivo PRIMERO.

```
procedimiento copia_arch_tex;

 arch1, arch2: Archivo_Texto;
 car: caracter

inicio
 Asociar (arch1, "Primero")
 Asociar (arch2, "Segundo")
 Abrir (A, arch1)
 Abrir (E, arch2)
 Leer (arch2, car)
 mientras no EOF(arch2)
        Escribir (arch1, car)
        Leer(arch2)
 fin mientras
 cerrar (arch1)
 cerrar (arch2)
fin
```

Cuadro 1.

Por último, nótese que este tipo de archivo tiene una organización estrictamente secuencial, debido a las operaciones que soporta, para acceder al carácter que se encuentra con respecto al principio del archivo en la posición n-ésima hay que acceder obligatoriamente a los n-1 caracteres que le preceden.

Archivos no tipificados

Al describir los archivos de texto se suponía que su estructura era una secuencia de caracteres, en la que podían aparecer varios delimitadores final de línea, en los archivos no tipificados no se realiza ninguna suposición acerca de la naturaleza de los datos contenidos en los archivos. Los archivos son tratados como meras secuencias de bytes. Estos archivos suelen utilizarse cuando las operaciones de entrada/salida son críticas durante la ejecución de determinadas tareas o cuando el tratamiento que se desea dar al archivo no necesita interpretar los datos contenidos en éste (por ejemplo, una rutina para la copia de archivos).

La declaración de un archivo no tipificado en seudocódigo tendrá la siguiente forma, *Ident*: **Archivo**. Además, este tipo de archivos también utilizarán los procedimientos **Asociar**, **Abrir**, y **Cerrar** de igual forma que los archivos de texto

pero la apertura está restringida a los modos de **entrada** y de **salida**, no siendo posible el modo **adición**.

Las instrucciones para lectura y escritura serán en seudocódigo las siguientes: **LeerBloque (Id_Arch, Id_Var, Tamaño, Cantidad)** y **EscribirBloque(Id_Arch, Id_Var, Tamaño)**. Los parámetros utilizados son los siguientes: **Id_Arch**, indica el identificador de archivo del que se leerá o en el que se escribirá, **Id_Var** (se utilizará una variable de tipo **Byte** o cuyo tipo base sea éste) indica la variable que recogerá o contendrá los datos según sea la operación de lectura o de escritura, respectivamente, el parámetro **Tamaño** (de tipo entero) indicará el número de bytes a leer o a escribir y por último el parámetro **Cantidad** (de tipo entero) indica la cantidad de datos realmente transferida, tomará el valor cero cuando se haya detectado el final de archivo.

El cuadro 2 muestra el seudocódigo para copiar dos archivos utilizando un acceso secuencial.

Los procedimientos **LeerBloque** y **EscribirBloque** permiten acceder de forma secuencial a este tipo de archivos, sin embargo, los archivos no tipificados también permiten el acceso aleatorio o directo mediante el procedimiento **Apuntar (Id_Arch, Pos_Bloque, Tamaño)**. Éste avanza el puntero asociado al archivo, indicado por el parámetro **Id_Arch**, hasta el bloque (que tendrá el tamaño indicado por el parámetro **Tamaño**) indicado por el parámetro de tipo entero, **Pos_Bloque**, una vez situado el puntero, se puede proceder a la lectura o escritura de dicho bloque. También dispondremos de un procedimiento que devuelve el tamaño de un archivo en bytes **Long_Arch(Id_Arch)**.

El cuadro 3 contiene el seudocódigo para copiar los bloques impares de un archivo no tipificado a otro.

Archivos tipificados

La principal ventaja que presenta el tratamiento que se da a un archivo no tipificado es la velocidad de acceso y el control que tiene el programador sobre la recuperación de los datos. El gran inconveniente es que si se desea interpretar los bytes recibidos para presentar información legible, se deben implementar procedimientos para interpretar los datos y presentar éstos como información. Los archivos tipificados resuelven este problema.

```
procedimiento Copia_binaria;

 arch1, arch2: Archivo
 buffer: array (100) Byte
 tam, cant: entero

inicio
  Asociar (arch1, "Primero")
  Asociar (arch2, "Segundo")
  Abrir (E, arch1)
  Abrir (S, arch2)
  LeerBloque (arch1, buffer, tam, cant)
  mientras cant <> 0
      EscribirBloque(arch2, buffer, cant)
      LeerBloque(arch1, buffer, tam, cant)
  fin mientras
  Cerrar (arch1)
  Cerrar (arch2)
fin
```

Cuadro 2.

Un archivo tipificado contiene datos de un tipo particular, manteniendo una estructura muy rígida que depende, y se define, a través del tipo de datos que contiene.

Se va a introducir el tipo de datos *registro* que normalmente está asociado al concepto de archivo tipificado. Un registro, es un tipo de datos que a su vez agrupa a un conjunto de datos de igual o distinto tipo, denominados campos, que están relacionados lógicamente.

```
procedimiento copia_impares
 arch1, arch2: Archivo
 tam, cant, tam_tot: entero
 buffer: array(100) Byte
 cont, num_reg: entero

inicio
 Asociar (arch1, "primero")
 Asociar (arch2, "segundo")
 Abrir (E, arch1)
 Abrir (S, arch2)
 tam = 100
 Tam_tot = Long_Arch (arch1)
 cont = Tam_tot / 100 + 1
 {la variable cont, contiene el número de
  bloques de 100 bytes que contiene el
  archivo a leer}
 num_reg = 1
  {num_reg indica la posición del registro
   a leer}
  mientras num_reg <= cont
      Apuntar (arch1, num_reg)
      LeerBloque (arch1, buffer, tam, cant)
      EscribirBloque (arch2, buffer, cant)
      num_reg = num_reg + 2
  fin mientras
  Cerrar (arch1)
  Cerrar (arch2)
 fin
```

Cuadro 3.

Para declarar en seudocódigo un registro basta con elegir un identificador y tras éste poner la palabra, que se considerará reservada, **registro** y tras ésta la lista de identificadores para los datos que contiene el registro indicando su tipo. Se podrá acceder a la totalidad del registro especificando el identificador de éste, el acceso a cada uno de sus campos se realizará poniendo, tras el identificador del registro, un punto y el nombre del campo. El cuadro 4 muestra entre otras cosas, la declaración de un registro, **Personal**, que está compuesto de dos campos de tipo cadena (**Nombre** y **Dni**) y un campo de tipo real (**Sueldo**).

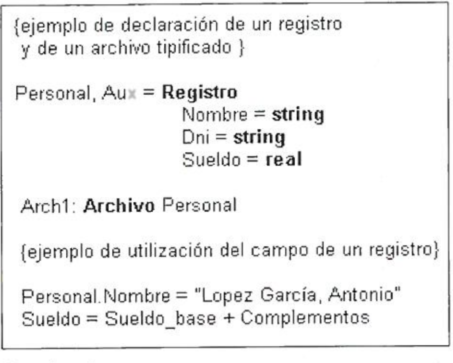
Para declarar un identificador de archivo tipificado, basta con poner tras éste la palabra reservada **Archivo** y a continuación el nombre del registro, que será la estructura básica del archivo.

Se utilizan de igual forma los procedimientos **Abrir**, **Cerrar**, **Asociar**, **Leer** y **Escribir**, la única diferencia estará en los dos últimos procedimientos, que junto al identificador de archivo, aceptarán una variable del mismo tipo que la del registro del archivo. Esta variable contendrá los datos leídos o los datos a escribir, tras la ejecución de alguna de estas dos instrucciones se producirá un desplazamiento hacia delante del puntero asociado al archivo.

El tipo de acceso que se podrá aplicar a estos archivos podrá ser secuencial o directo. En este último caso se utilizará el procedimiento **Apuntar** y la función **Long_Arch.** El procedimiento **Apuntar** admitirá como primer parámetro el identificador del archivo tipificado y como segundo parámetro una variable de tipo entero, el efecto será situar el puntero asociado al archivo en el registro cuya posición se indica en la variable de tipo entero. En cuanto a la función **Long_Arch**, admitirá un único parámetro, el identificador de archivo, y devolverá el número de registros que contiene el archivo.

Los modos de apertura que estos archivos permiten son el de **Entrada** y el de **Salida**.

El cuadro 5 muestra un programa para el cálculo de la media de los sueldos en un archivo con tipo de datos **Personal**, este programa ilustra el trata-



```
{ejemplo de declaración de un registro
 y de un archivo tipificado }

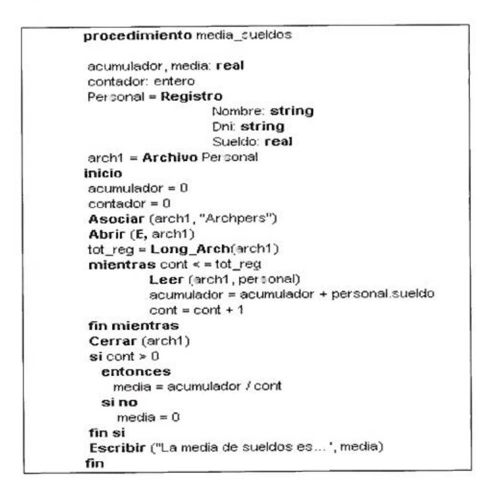
Personal, Aux = Registro
                    Nombre = string
                    Dni = string
                    Sueldo = real

Arch1: Archivo Personal

{ejemplo de utilización del campo de un registro}

Personal.Nombre = "Lopez García, Antonio"
Sueldo = Sueldo_base + Complementos
```

Cuadro 4.



```
procedimiento media_sueldos

 acumulador, media: real
 contador: entero
 Personal = Registro
                 Nombre: string
                 Dni: string
                 Sueldo: real
 arch1 = Archivo Personal
inicio
 acumulador = 0
 contador = 0
 Asociar (arch1, "Archpers")
 Abrir (E, arch1)
 tot_reg = Long_Arch(arch1)
 mientras cont < = tot_reg
         Leer (arch1, personal)
         acumulador = acumulador + personal.sueldo
         cont = cont + 1
 fin mientras
 Cerrar (arch1)
 si cont > 0
   entonces
     media = acumulador / cont
   si no
     media = 0
 fin si
 Escribir ("La media de sueldos es... ', media)
fin
```

Cuadro 5.

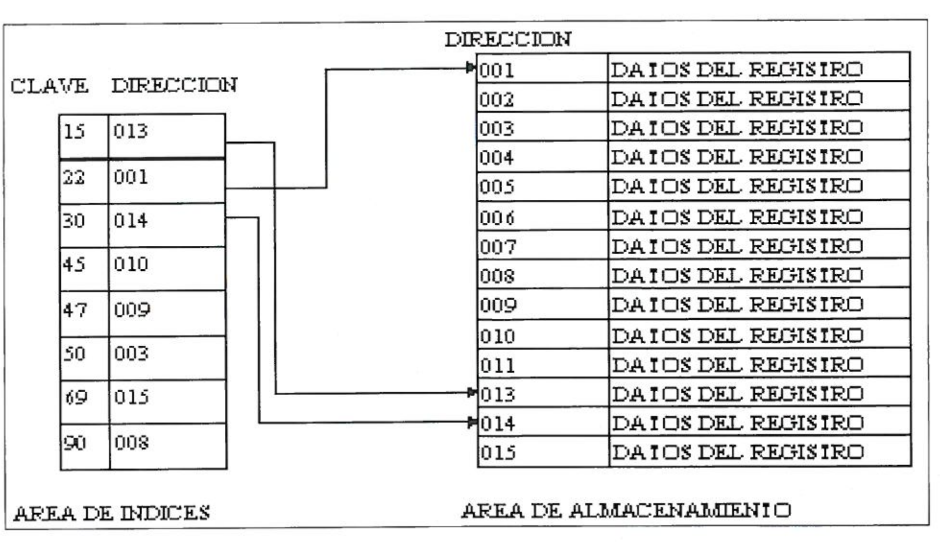
miento secuencial que se puede dar a un archivo tipificado.

El cuadro 6 muestra un programa que solicita la entrada de números comprendidos entre 1 y el tamaño de un archivo con tipo de datos **Personal**, mostrando la información contenida en el registro de posición dicho número. Este programa ilustra el acceso directo a estos archivos.

```
procedimiento acceso_directo
 personal : registro
                  Nombre : string
                  Dni: string
                  Sueldo: entero
Arch1: Archivo Personal
posicion = entero

inicio
 Asociar (arch1, "Archper")
 Abrir (E, arch1)
 Escribir ("Introducir un número de registro ")
 Leer (posicion)
 mientras posicion <> 0 {salir cuando el usuario teclee 0}
      si (posicion < 0) o Long_Arch (arch1) < posicion
         Escribir ("No existe ese registro")
      si no
         Apuntar(arch1, posicion)
         Leer (arch1, personal)
         Escribir ("DNI: ", personal.dni)
         Escribir ("Nombre: ", personal.nombre)
         Escribir ("Salario; ", personal.salario)
      fin si
      Escribir ("Introducir un número de registro ")
      Leer (posicion)
 fin mientras
 Cerrar(arch1)
fin
```

Cuadro 6.



Cuadro 7.

OTROS TIPOS DE ARCHIVOS

Como es fácil observar, la recuperación de la información contenida en un archivo perteneciente a cualquiera de los tipos anteriores es adecuado si se desea acceder a la información bien secuencialmente o bien mediante la posición que ocupan los datos respecto al inicio del archivo. Sin embargo, la búsqueda de una determinada información sería un proceso muy costoso en operaciones de E/S, y por tanto en tiempo, pues exigiría la inspección secuencial de cada uno de los registros del archivo. Existen diversas organizaciones "lógicas" para archivos que facilitan la búsqueda de información, este tipo de organización no se corresponde con el tipo de acceso físico real que seguirá siendo secuencial o directo.

Para resolver el problema de acceder a la información correspondiente a un determinado valor (clave) se utilizan los **archivos indexados.**

ARCHIVOS INDEXADOS

Estos archivos están presentes en todos los lenguajes de programación orientados a gestión y en los lenguajes de manipulación de bases de datos. Definiremos el concepto de clave principal, como una combinación de campos que identifican a un registro de manera unívoca. Por ejemplo, en el registro **Personal** del cuadro 4 el campo **Dni** seria una clave principal si no se permitiese la existencia de dos registros con idéntico valor en **Dni**.

Los archivos indexados constan de un área de índices y de un área de almacenamiento. El primero consta de pares de valores con cada uno de los diferentes valores de las claves presentes en el área de almacenamiento y el número de registro donde se encuentra almacenada realmente la información. Éste área se suele encontrar ordenada por la clave para facilitar la búsqueda. La totalidad de la información se encuentra en el área de almacenamiento. Cuando se desea acceder a un determinado registro mediante su clave, primero se localiza la situación del registro en el área de índices y si se ha encontrado, se dispone de su dirección en el área de almacenamiento, con lo que la totalidad de la información puede ser recuperada.

Se puede utilizar el símil de un libro que disponga de un índice temático. En éste se encuentran las palabras "clave" ordenadas alfabéticamente, junto a la palabra se encuentra el número de la página donde se encuentra la totalidad de la información referente a dicha palabra clave, una vez localizada la palabra clave, se procede a la lectura de la página indicada. El cuadro 7 muestra la idea en la que se basan los archivos indexados.

Para implementar este tipo de archivos en lenguajes que no dispongan de archivos indexados es importante seleccionar una buena estructura de datos para poder aplicar sobre los índices un algoritmo de búsqueda eficiente. Normalmente la estructura seleccionada es la de un árbol B+ junto con un algoritmo de búsqueda binaria.

Además de las operaciones usuales de lectura y escritura, se dispone de la operación de lectura de un registro dada una clave y la inserción de registros.

EL SISTEMA DE FICHEROS UNIX

Fernando J. Echevarrieta

El sistema de ficheros constituye una parte fundamental en todo Sistema Operativo. Continuando con la perspetiva del programador/administrador de UNIX iniciada el pasado mes, este será el tema del presente artículo.

El término *sistema de ficheros* presenta una cierta ambigüedad al emplearse indistintamente para referir dos elementos íntimamente relacionados. Por una parte, denomina al "conjunto software dentro de un S.O. que se encarga de la creación, almacenamiento, recuperación, protección y gestión en general de la información sobre un soporte físico accesible on-line". Por otra, cualifica al medio físico soporte. En el presente artículo se explica el sistema de ficheros UNIX que, como se verá, tiene algunas particularidades especialmente interesantes.

Para UNIX, un fichero es una secuencia de bytes de la que no es necesario declarar cuanto espacio ocupa. El S.O. no proporciona ninguna estructura adicional como límites de registro, bloques de ficheros, ficheros indexados o tipados. Por ello, el tratamiento del sistema de ficheros a estos niveles será labor de las aplicaciones.

PARADIGMA UNIX DE E/S

Como de todos es sabido, UNIX es un S.O. de tiempo compartido y monoprocesador. Se trata de un sistema orientado al proceso por lo que los programas de aplicación son procesos que se encuentran a nivel de usuario y pueden comunicarse e interactuar con el kernel a través de system calls.

Como se ha venido indicando una y otra vez a lo largo de esta serie de artículos, UNIX unifica todos los procesos de entrada y salida (en adelante E/S) y considera todo componente del mundo real como un fichero. El paradigma de E/S en UNIX define cuatro primitivas generales para tratar con ficheros y, por tanto, con todo el universo: *open, read, write* y *close.*

La llamada *open* retorna un descriptor de fichero. Por ejemplo:

fdesc=open("fichero",O_CREAT|O_RDWR,0644)

Sobre este descriptor (o cualquier otro generado por otra llamada, por ejemplo, *socket*), podrán actuar *read y write,* que siempre constan de tres argumentos, el descriptor sobre el que actúan, la dirección de un buffer de entrada o salida según proceda y el número de bytes a ser leído o escrito. Por ejemplo:

n=read(fdesc,buff,24)

Ambas llamadas realizan la transferencia desde o hasta la posición "actual" actualizándola, por lo que es común emplearlas en una estructura *while* (mientras haya datos, por ejemplo). En caso de que se intente leer más de lo que hay, read leerá lo que queda y devolverá el número de bytes leído. Una lectura posterior devolvería 0.

Por último, *close(descriptor)* cerrará el descriptor que en cualquier caso será cerrado por el sistema al finalizar el programa aunque éste no haya realizado la llamada a *close.*

MODO Y BITS ASOCIADOS A UN FICHERO

Como ya se expuso en su día, cada fichero se encuentra asociado a un owner o propietario que queda identificado por un número denominado UID. Este número se almacena junto a la información que forma el fichero. También se comentó que para permitir compartir ficheros, UNIX proporcionaba un GID o identificador de grupo para cada fichero, pudiendo un usuario pertenecer a varios grupos pero un fichero sólo a uno de ellos. Para el control de acceso de las aplicaciones, el S.O. compara los UID y GID de las mismas con las del fichero y de esa forma determina si debe ser permitido o no. De esta forma, se permite controlar el acceso de la misma forma al propietario,

grupo y al resto a través del modo del fichero representado por nueve bits de protección (figura 1).

El sistema de ficheros se encuentra organizado según una jerarquía de directorios, a los cuales también se aplican los nueve bits de protección. En el caso de los directorios, el permiso de lectura permite ver la lista de ficheros que contienen; el de ejecución permite la entrada en él (referencia a un fichero dentro del directorio); y el de escritura, insertar o eliminar ficheros.

En UNIX, existen más bits asociados a cada fichero, como se muestra en la misma figura 1. Así, otros tres bits: B, A y 9, denominados *set UID bit, set GID bit* y *sticky bit*, respectivamente, controlan la "forma" de ejecución del programa. Si el primero de ellos se encuentra activado, el proceso generado al ejecutar el fichero adoptará los permisos del propietario del fichero en lugar de los del propietario del proceso (el sujeto que lo ha lanzado). De esa forma se produce una herencia de UID (usuario). Análogamente, el segundo produce una herencia de GID otorgando al proceso los permisos del grupo del propietario del fichero. La "personalidad" que adopta el proceso cuando existe esta herencia de propiedades es lo que se denomina *usuario o grupo efectivo*, según proceda. El tercero de los bits, el sticky o "pegajoso" se creó en un principio para obligar al sistema a que mantuviera el contenido del fichero en memoria, generalmente por tratarse de ficheros de uso generalizado y con el objeto de acelerar su arranque evitando su carga desde disco y, por tanto, sólo puede ser activado por el root. Hoy en día y con los nuevos sistemas de jerarquías de memoria no tiene mucho sentido. Estos tres bits pueden ser modificados mediante el comando *chmod modo fichero*, que se explicó con detalle en el número 6 de la revista. Para ello, bastará añadirlos a la codificación del modo en octal. Así, los modos *1000*, *2000* y *3000* corresponderán a la activación de los bits "pegajoso" y de herencia de grupo y usuario respectivamente y, como los demás modos, podrán ser sumados modos correspondientes a los bits de permiso. Ejemplo:

chmod 2755 fichero

colocará herencia de grupo al fichero

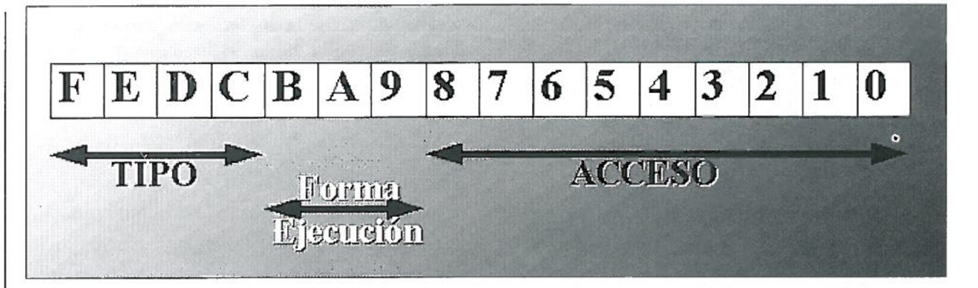


Figura 1. Bits asociados a un fichero.

(2000) y le dará todos los permisos al propietario (0700) y sólo los permisos de lectura y ejecución a todos los demás usuarios (grupo, 0050 y resto 0005).

VOLÚMENES DE UN SISTEMA DE FICHEROS

El sistema de ficheros global UNIX se encuentra formado por varios *volúmenes* o sistemas de ficheros autónomos físicamente. De estos existen dos tipos, el *raíz*, siempre presente, y los demás, que pueden ser montados y desmontados. Mediante esta característica y el sistema de nombres de UNIX se consigue que los usuarios "imaginen" la jerarquía del sistema de ficheros quedando oculta para ellos la estructura real. La idea principal es no forzar al usuario a tener que identificar dispositivos físicos como en otros S.O. como DOS en los que es necesario indicar el disco duro, floppy, o CD-ROM donde se encuentra un fichero o directorio o cambiar a él antes de acceder a sus ficheros. En UNIX es el administrador el que incorpora las "jerarquías" de un dispositivo a otro a base de montar los volúmenes correspondientes al disco duro, floppy, CD-ROM, red, etc, al punto deseado de la jerarquía. Para ello, elige una jerarquía como raíz y crea en ella un directorio vacío. Después indica al S.O. que desea solape la jerarquía del volumen a montar con el directorio. A partir de este momento, el S.O. enlazará los nombres dando la impresión de un único sistema. También es posible dividir un único dispositivo físico en distintos volúmenes, como ocurre con las particiones de un disco. Esto se suele hacer debido a que, aunque el sistema de ficheros se verá como un todo, en caso de catástrofe sólo se perderá el volumen afectado y no todo el conjunto. Por otra parte, también se garantiza de este modo un tamaño para cada volumen ya que si otro se llena, no sobrepasará sus límites para seguir creciendo en éste.

La acción de montar y desmontar volúmenes se realiza mediante los comandos *mount* y *umount* respectivamente y sólo podra ser llevada a cabo por el root, si bien cualquier otro usuario podrá ejecutar mount para ver qué volúmenes se encuentran montados en el sistema. La sintaxis varía de un sistema a otro por lo que el lector deberá acudir al manual en línea. Como ejemplo, en Linux:

mount /dev/hda3 /home

montaría la partición 3 del primer disco duro en el directorio /home y

mount -t umsdos /dev/hdb2 /dosd

montaría la partición 2 del segundo disco duro en el directorio /dosd gestionándola como un sistema *umsdos*, es decir aplicando un sistema unix sobre una partición con formato DOS.

En ocasiones no es necesario especificar más que el nombre del volumen o del punto de anclaje. Por ejemplo:

mount /dosd.

Esto ocurre cuando ese volumen se encuentra reflejado en la base de datos de volúmenes del sistema, que se encuentra en el fichero */etc/fstab* cuyo formato se puede apreciar en el listado 1. Al arrancar el S.O. una de las

FICHERO /etc/fstab de Linux

/dev/hdb3	swap	swap	defaults
/dev/hdb2	/	ext2	defaults
/dev/hda1	/dosc	msdos	defaults
/dev/hdb1	/dosd	umsdos	defaults
/dev/hda5	/dose	msdos	defaults
/dev/sbpcd	/cdrom	iso9660	ro
none	/proc	proc	defaults

Listado 1.

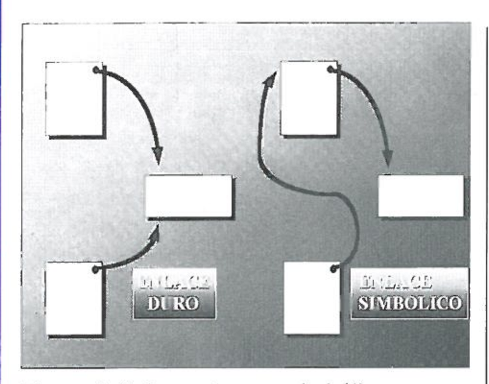


Figura 2. Enlaces duros y simbólicos.

primeras cosas que hace es leer este fichero y montar los volúmenes adecuados.

Una consideración a tener en cuenta a la hora de escribir en este fichero, lo que sólo se podría hacer como root, es el orden en que figuran los volúmenes. Así para montar un volumen, es necesario que el punto de anclaje exista, es decir, que se haya montado previamente. Por ello si en el ejemplo se colocara en el fichero el volumen */dev/hda5* antes que el */dev/hdb2*, correspondiente al raíz, la primera no se montaría ya que antes de montar el volumen raíz no existe el directorio */dose*. Sin embargo una vez arrancado el sistema, sí bastaría con teclear:
mount /dose

Siguiendo con el ejemplo de Linux, como primera medida, y por si acaso, conviene hacer un backup del fichero antes de modificarlo ya que si se configura de manera errónea, podría no arrancar el sistema. Si esto le ocurre, arranque desde disquete (el volumen raíz será la raíz del disquete), monte la partición que debe ser raíz del disco duro a mano en un directorio del disquete (asumamos en el disquete un directorio /mnt):
mount /dev/hdb2 /mnt
y restaure el fichero original. Por ejemplo:
mv /mnt/etc/fstab.bak /mnt/etc/fstab

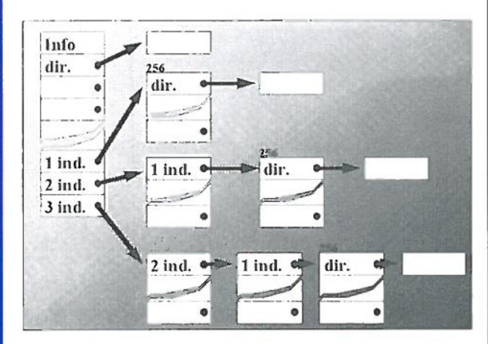


Figura 3. Localización de bloques físicos.

Como se puede ver, el nombre y path de los ficheros depende de los puntos de anclaje de los volúmenes que los contienen.

TIPOS DE FICHEROS

Los ficheros para UNIX se pueden clasificar en cuatro grandes tipos: ordinarios, directorios, enlaces y ficheros especiales y, dentro de este último tipo se engloban todos los demás mecanismos de entrada/salida: dispositivos, pipes con nombre y sockets de dominio unix.

FICHEROS ESPECIALES

UNIX garantiza la total compatibilidad entre los mecanismos de entrada y salida de ficheros y de cualquier tipo de dispositivos o medio de comunicación entre procesos siguiendo el paradigma expuesto anteriormente. Es por ello que es necesario definir clases de ficheros, definiendo una serie de ficheros "especiales". La mayoría de estos ficheros son ficheros asociados a dispositivos y se suelen encontrar (por convenio) en el directorio */dev* (devices, dispositivos). Los dispositivos pueden ser de dos tipos: de *bloques* (con buffer) o de *caracteres* (sin buffer). Generalmente su propietario es el root, aunque existen excepciones como para los terminales */dev/tty** cuyo propietario es el usuario mientras dure la sesión. Debido a que estos ficheros representan dispositivos físicos de la máquina, de nuevo será el root el único usuario autorizado para su creación. Su importancia es tal que de nada sirve tener un ratón o un lector de CD-ROM si no se han creado los ficheros asociados, debido a que el S.O. no dispondrá de mecanismo alguno para dialogar con este hardware.

Para la creación de estos ficheros, el S.O. provee el comando *mknod* que se emplea de la forma:
mknod nombre [b|c] número_mayor número_menor

donde *b* o *c*, opcionales, especifican si el fichero va a ser de bloques o caracteres y el dispositivo en cuestión queda identificado por los números. Todos los dispositivos se clasifican en grupos, por ejemplo, impresoras, lectores de CD-ROM, tarjetas de sonido, y a cada grupo se le asigna un *número_mayor* identificativo. Dentro de estos grupos, cada dispositivo concreto recibe un *número_menor*, por ejemplo CD-ROM IDE, CD-ROM controlable SoundBlaster, CD-ROM SCSI. El problema surge al buscar los números adecuados. Algunos sistemas UNIX los definen en el fichero */etc/conf*. No es el caso de Linux, dónde residen en el fichero */usr/src/linux/include/linux/major.h*. La identificación de los dispositivos suele ser un quebradero de cabeza aunque en la mayoría de los casos, la distribución del sistema incorpora una utilidad a tal efecto. Así, en Linux, se facilita el script *MAKEDEV* que se puede encontrar en el directorio */dev*.

Si lo que se desea es crear este tipo de ficheros desde programa, la API del núcleo facilita la función:
*int mknod(char *nombre, mode_t modo, dev_t dispositivo)*

Otros dos tipos de ficheros especiales son los *pipes con nombre* (named pipes) o *ficheros FIFO* y los *sockets de dominio UNIX*. Ambos suponen un mecanismo de comunicación entre procesos que se presenta como un fichero en el sistema. Se les puede distinguir mediante la opción -F del comando ls que ya se explicó en su momento. En el capítulo dedicado al IPC de sockets de Berkeley se estudiará el segundo tipo, por lo que en este momento sólo se hablará de los ficheros FIFO. Como su nombre indica se trata de ficheros-cola de mensajes (First In-First Out). Cuando dos procesos desean comunicarse, abren el fichero FIFO como si se tratara de uno normal. El productor escribirá en él y el consumidor leerá de él.

Para generar un fichero FIFO, se puede utilizar también la orden *mknod*, mediante
mknod nombre p
aunque existe la orden específica
mkfifo nombre

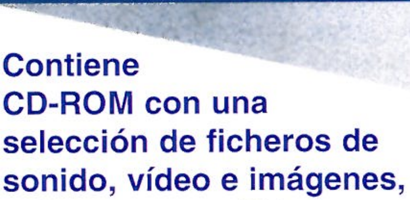
ENLACES DUROS Y BLANDOS (SIMBÓLICOS)

UNIX separa la información de un fichero de su entrada en un directorio. Así pues, permite tener varias entradas de directorio para un sólo fichero. Cada una de estas entradas recibe el nombre de *enlace duro* (hard link). La ventaja de esto es que un fichero puede aparecer en diversos lugares sin ocupar espacio físico más que una sóla vez. Por otra parte,

LOS LIBROS DE INFORMÁTICA MÁS ACTUALES AL MEJOR PRECIO











Contiene CD-ROM con una selección de ficheros de sonido, vídeo e imágenes, así como las utilidades disponibles actualmente para Windows 95.

Contiene CD-ROM con 800 programas de acceso rápido a INTERNET







CADA LIBRO POR SÓLO:

1.995 ptas.
IVA INCLUIDO



Solicite catálogo al Tel. (91) 741 26 62

la actualización en un lugar implica la actualización en todos ya que se trata de un fichero único se use el nombre que se use, por lo que la información siempre será consistente.

Un enlace duro es, por tanto, indistinguible del original. No se trata de que sean algo distinto pero imposible de distinguir desde fuera, sino que ahora, como muestra la figura 2, el fichero posee dos (o más) entradas de directorio y "tanto monta, monta tanto". La única restricción para los enlaces duros, que se comprenderá al estudiar la implementación física del sistema de ficheros es que no puede atravesar volúmenes. Es decir, todos los enlaces deben residir en el mismo volumen que los datos. Debido a la existencia de enlaces duros, el sistema de fichero UNIX deja de ser un árbol para convertirse en un *grafo acíclico*.

Para crear o eliminar enlacer duros, el S.O. proporciona las funciones:

*int link(char *nombre, char *nombre_enlace)*
*int unlink(char *nombre)*

Cuando se ejecuta *unlink* o la orden *rm*, se borra el enlace (su entrada en el directorio), pero el fichero no desaparece hasta que todos sus enlaces duros han sido borrados.

El comando del sistema que crea enlaces, ln, se puede emplear de la forma:

ln fichero
ln fichero enlace
ln lista_ficheros directorio

Un enlace simbólico, por contra, no es más que un fichero especial que contiene el nombre de otro fichero, como se puede ver en la figura 2. Es decir, en lugar de apuntar a los datos del fichero, apunta a una de las entradas del directorio del mismo, es decir, a uno de sus enlaces duros. Como ventaja, permite apuntar a cualquier fichero aunque resida en otro volumen. El inconveniente principal es que no es fiable ya que puede llamar a un objeto que no existe; situación que se da cuando la entrada apuntada es eliminada, ya que el enlace simbólico continuará conteniendo el nombre de la misma. Otro problema es que debido a la existencia de enlaces simbólicos, el sistema de ficheros UNIX deja de ser un grafo acíclico para convertirse en un *grafo cíclico*, lo que puede dar lugar a la existencia de bucles. El trivial, el de un enlace que se apunta a sí mismo. Todas las acciones sobre un enlace simbólico se realizan sobre el fichero destino salvo *ls* y *rm*.

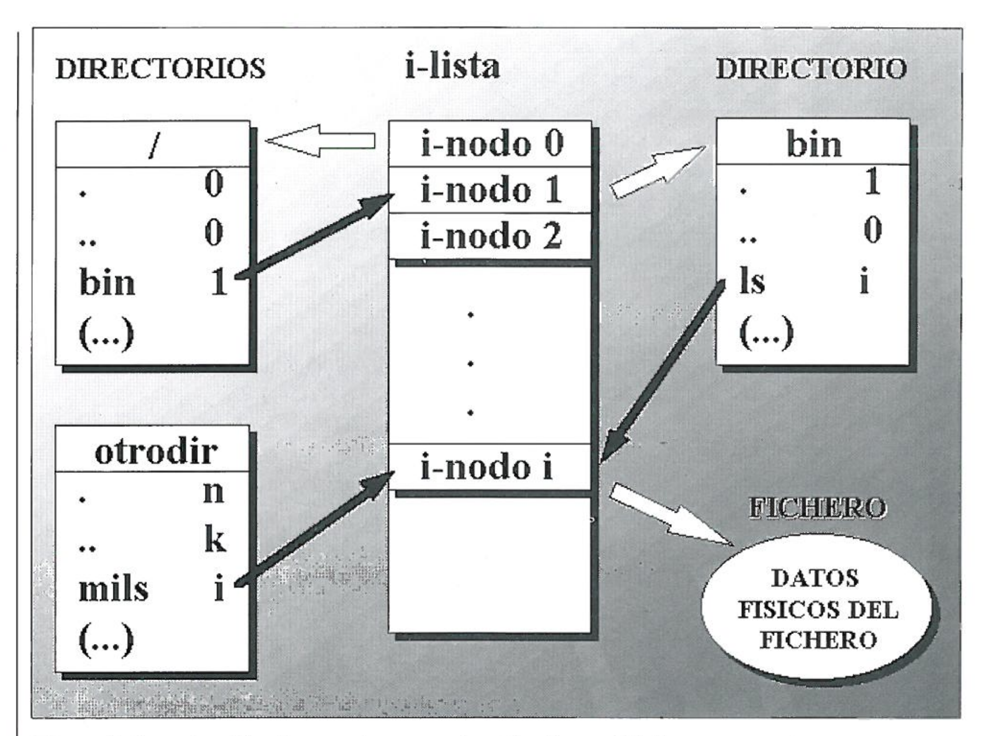


Figura 4. Organización de un volumen en función de una i-lista.

Para generar enlaces simbólicos desde programa, la API del núcleo proporciona la función: *int symlink(char *nombre, char *nombre_enlace)*

Para generar enlaces simbólicos desde una shell se emplea también el comando *ln*. Esta vez, con la opción *-s*.

IMPLEMENTACIÓN FÍSICA: I-NODOS. STAT.

Como se ha mencionado, un fichero se puede asimilar a una cadena de bytes que se almacenan en bloques de datos en un medio físico. El bloque del UNIX original era de 512 bytes siendo el de la variante de Berkeley de 4Kb aunque se permite que el último bloque de un fichero sea más pequeño. Este es el significado del tamaño de bloque en el sistema de ficheros por el que pregunta el programa de instalación de Linux (y los de muchos otros UNIX).

La información sobre un fichero y las direcciones de los bloques se encuentran en estructuras denominadas *i-nodos* (i=índice) que se agrupan en *i-listas*. Los i-nodos almacenan información sobre el fichero en soporte estable; información como propietario, grupo, bits de modo, momentos de último acceso y última modificación, tamaño, disco físico y volumen, número de entradas de directorio asociadas (sobre esto se volverá más adelante), número de bloques de disco usados y tipo básico del fichero. En la figura 3 se puede observar como el S.O. alcanza los bloque físicos de un fichero a partir de su i-nodo.

Será posible acceder a la información presente en un i-nodo mediante la llamada al sistema:

stat(nombre_fichero,&resultados)

donde *&resultados* es la dirección de un área de memoria en la que debe caber una estructura stat similar (dependiente de versión) a la que se puede ver en el listado 1. El uso de esta llamada no precisa ningún tipo de permiso (lectura, escritura, ejecución) sobre el fichero, pero sí sobre su ruta de acceso. Si se aplica stat sobre un enlace simbólico, se obtendrá la información sobre el fichero apuntado por éste. Si se desea la información del propio enlace será necesario emplear:

lstat(nombre_fichero,&resultados)

Cada volumen, posee su propia i-lista agrupando todos sus i-nodos. Un directorio no es otra cosa que un fichero-tabla que contiene una entrada por cada fichero, conteniendo el nombre del fichero y un puntero denominado *i-número* al i-nodo correspondiente.

En la figura 4 se puede observar una hipotética i-lista en la que el *i-nodo 0* apunta al directorio raíz. En él, el directorio . contiene una referencia al *i-nodo 0*, es decir, señala a sí mismo. Y lo mismo ocurre con el directorio, ya que se trata del directorio raíz y no existe anterior. Sin embargo, el i-número asociado al fichero *bin* apunta al *i-nodo 1*. Se puede ver que el *i-nodo 1* apunta a otro directorio, por tanto *bin* es un directorio. Siguiendo el sistema de punteros, se ve como, en efecto, . apunta a sí mismo y .. al directorio raíz. En cambio, *ls* apunta al *i-nodo 12* que esta vez sí remite a un fichero físico.

ACCESO CONCURRENTE A FICHEROS

Un descriptor de fichero apunta a una estructura de datos que, entre otras informaciones, almacena la posición actual en el fichero. Sin un proceso realiza una llamada *fork*, tanto el padre como el hijo compartirán el mismo descriptor de fichero que apuntaría a una única estructura por lo que, si uno varía la posición de acceso al fichero, ésta variará también para el otro proceso.

Por el contrario, cada llamada a la función open devuelve un descriptor de fichero distinto. Así, si dos procesos realizan un open sobre un mismo fichero, cada uno obtendrá un descriptor distinto y podrán acceder al fichero de manera independiente. Aquí surge el problema de que UNIX no provee exclusión mutua por lo que el fichero contendrá lo último que se haya escrito sobre él. Así pues, será responsabilidad del programador garantizar la corrección de los resultados. Para facilitar esta labor, es posible el uso de *locks* o *cerrojos* sobre los ficheros que según el tipo de UNIX vendrán facilitados en unas ocasiones mediante llamadas al sistema como *lockf* o *flock* y en otras mediante flags para la llamada open, como *O_EXCL*.

OTRAS FUNCIONES DE LA API

Para concluir el artículo se mostrarán algunas otras llamadas al Sistema de Ficheros, complementarias a las que se han visto:

Acceso a directorios

*DIR *opendir(char *name)*

Abre un directorio y le asocia un flujo

*struct dirent *readdir(DIR *dir_desc)*

Devuelve puntero a la siguiente entrada de directorio, que se define (dependiente de versión) según una estructura como:

```
struct dirent {
 long   d_ino;  /* Número de i-nodo */
 unsigned short  d_reclen; /* Longitud entrada */
 unsigned short  d-namlen; /* Longitud d_name */
char      d_name[NAME_MAX+1];
};
```

*closedir(DIR *dir_desc)*

Cierra el flujo y libera el descriptor.

Creación de ficheros

*FILE *fopen(char *nombre,char *modo_acceso)*

Abre un fichero. Donde *modo_acceso*: r (lectura), w (escritura), a (añadir). Ejemplo: fp=fopen(fich_datos,"r")

*int creat(char *nombre, int mode)*

Equivale a *open(nombre,(O_CREAT|O_TRUNC),modo)*

El modo es modificado por umask según la formula (~umask) & modo. El umask por defecto es 022.

int umask(int mascara)

Asigna la nueva máscara de fichero y devuelve la antigua

Creación/borrado directorios

*int mkdir(char *nombre,int modo)*

*int rmdir(char *nombre)*

el directorio debe estar vacío. Retorna 0 o -1

Cambio de propietario

*int chown(char *nombre, uid_t propietario, gid_t grupo)*

int fchown(int fd, uid_t propietario, gid_t grupo)

El parámetro al que se le dé valor -1 no variará.

Atención: El usuario 0 es el root.

Cambio de modo

*int chmod(char *nombre, int modo)*

int fchmod(int fd, int modo)

Acceso directo

int lseek(int fd, int offset, int origen)

Donde el *origen* es: 0: principio, 1: posición actual, 2: fin y *offset* puede ser positivo o negativo. Es incorrecto hacer *lseek* sobre un pipe o socket. Si se usa lseek fuera del fichero y se escribe, el S.O. extenderá el fichero devolviendo 0 cuando acceda a una de las posiciones intermedias. En realidad, estas no ocuparán lugar, por lo que el tamaño representa el byte más alto accedido .

Consulta permisos

*int access(char *nombre, int modo)*

donde el modo es una combinación OR de R_OK (permiso lectura), W_OK (permiso escritura), X_OK (permiso ejecución) y F_OK (existencia). Devuelve 0 o -1.

Directorio actual

*char *getwd(char *name)*

Cambio directorio actual

*int chdir(char *name)*

ESTRUCTURA STAT (antigua)

```
struct stat {
        unsigned short st_dev;     /* Disp del i-nodo */
        unsigned short st_ino;        /* Número de este i-nodo */
        unsigned short st_mode;    /* Tipo fichero y modo */
        unsigned short st_nlink;  /* Número enlaces duros*/     unsigned short st_uid;    /* UID del propietario */
        unsigned short st_gid;    /* GID del propietario */
        unsigned short st_rdev;    /* Tipo de dispositivo */
        unsigned long  st_size;    /* Tamaño total */
        unsigned long  st_atime;   /* Último acceso */
        unsigned long  st_mtime;   /* Última modificación */
        unsigned long  st_ctime;   /* Último cambio status */
};
```

Tabla 1.

LA TRANSFORMADA DISCRETA DE FOURIER (PARTE II)

Juan Ramón Lehmann

En el presente capítulo se verá el teorema de la convolución en el dominio de la frecuencia, las aplicaciones reales de la IFT y la DFT y diseño de filtros en el dominio de la frecuencia.

Teorema de la convolución en el dominio de la frecuencia

En el número 10 de esta revista se trato brevemente los aspectos básicos del teorema de la convolución, así como en el articulo anterior. Sin embargo, este teorema, tiene un significado mucho más importante.

La correlación entre dos imágenes cualesquiera, no es más que la convolución de dos señales de las mismas

Supongamos un impulso cualquiera, un delta de Dirac perfecta, localizada perpendicular al eje del sistema en las coordenadas (x,y). La PSF (función respuesta característica del sistema) determinará una función de distribución en el plano de la imagen definida por d(x,y,x2,y2), donde x2 e y2 corresponden a las coordenadas de los puntos de la distribución **d** en el plano imagen.

Si en lugar de un punto se considera una distribución de ellos, la respuesta total del sistema i(x2,y2), suponiendo que el mismo sea lineal, será la suma de cada una de las PSF de cada punto de la distribución.

Matemáticamente podemos expresarlo:

$$i(x,y) = \int_{x_1=-\infty,\ y_1=-\infty}^{\infty} d(x_1,y_1,x,y)\cdot o(x_1,y_1)\,dx\,dy$$

Figura 1.

donde i(x,y) corresponde al valor de la distribución imagen en el punto (x,y) y o(x,y) con la del objeto en el plano origen.

El teorema de la convolución dice que si i(x,y) es la imagen resultante de convolucionar un objeto o(x,y) con una función respuesta al impulso d(x,y), la transformada I(u,v) puede también calcularse como el producto de las transformadas del objeto y la PSF.

Aplicaciones de la Transformada de Fourier

Filtrado en módulo y fase

Si la operación de filtrado además de hacerla por su parte real y por su parte imaginaria la realizamos por su módulo y fase, conseguimos la implementación de dos operaciones de forma automática, la multiplicación de

En el capítulo anterior se dio un pequeño repaso sobre algunos fundamentos a tener en cuenta antes de meterse de lleno en la Transformada de Fourier.

una imagen por una constante y su traslación en el espacio.

El producto por la constante se realiza haciendo que el módulo del filtro tome un valor distinto del unitario, mientras que el desplazamiento se debe a una fase lineal de pendiente constante. La magnitud del desplazamiento, en x e y, es proporcional a la inclinación de la fase en cada una de las dos coordenadas.

En la figura 2 se puede ver una representación gráfica de lo anteriormente mencionado. El módulo en cada punto vendría a ser el grosor del medio por el cual la imagen se transmite (que representaría la atenuación de la misma) y la fase lineal en cada punto sería el prisma en el cual la imagen se refleja o desplaza según la magnitud del ángulo de refracción de dicho prisma. Esto se puede contemplar claramente en la figura 2.

señales se relaciona con la convolución de las mismas de la siguiente manera:

$$f(x,y)\otimes g(x,y)=\int_{-\infty}^{\infty} f(\alpha)\cdot g(x-\alpha)\,d\alpha$$

$$f(x,y)\star g(x,y)=\int_{-\infty}^{\infty} f^{*}(\alpha)\cdot g(x+\alpha)\,d\alpha$$

Figura 3.

De acuerdo con las propiedades de la transformada de Fourier, podremos deducir que:

$$f^{*}(x,y)\cdot g(x,y) \Rightarrow F(u,v)\star G(u,v)$$

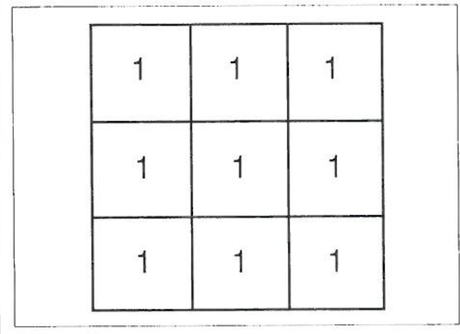
Figura 4.

donde F(u,v) y G(u,v) son la transformada de Fourier respectivamente de las dos señales de entrada f(x,y) y g(x,y).

El diseño de filtros, aunque tocado brevemente en el número anterior, es el objetivo de este apartado. Se explican en este punto una serie de filtros perfectamente aplicables a la teoría de la convolución lo cual es extensivo a la DFT.

SUAVIZADO

El filtro de la media, calcula el valor de un pixel en base a la media de los que le rodean. Este efecto se logra convolucionando con la máscara:



1	1	1
1	1	1
1	1	1

Figura 5.

Variando el valor central de la máscara, se consigue suavizar el suavizado o bruscalizarlo. Si por ejemplo se diese un valor más alto al valor central, el filtro paso bajo presentaría una frecuencia de corte mayor y menos abrupta.

La máscara a aplicar sería la de la figura 6.

En la cual m será superior a la unidad y tan alto como frecuencia de corte se desee.

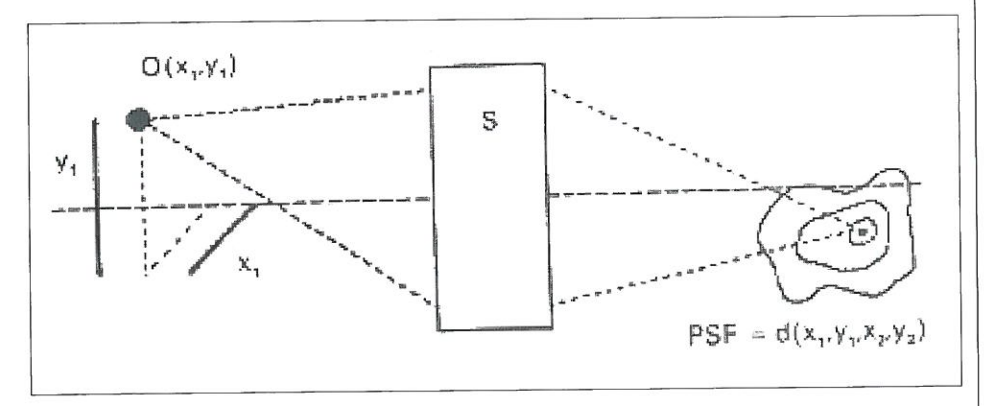


Figura 2.

CORRELACIÓN

El principal interés de la correlación es el seguimiento de objetivos y la búsqueda de patrones de imagen. El problema del seguimiento de objetivos radica en localizar el punto de la segunda imagen en el cual más se parece a la primera. Este punto se puede localizar mediante el valor máximo del mapa o(x,y) que resulta de correlacionar las dos imágenes.

La correlación entre dos imágenes cualesquiera, no es más que la convolución de dos señales de las mismas.

Aplicando a un caso continuo unidimensional, la correlación de las dos

De acuerdo con lo mencionado, se puede correlacionar dos imágenes en base a sus transformadas de Fourier, conjugando una de ellas y multiplicando. De la transformada inversa de Fourier que resulta (IFT) se desprende un mapa de correlación que marca de acuerdo a un máximo global, el desplazamiento relativo de ambas imágenes, siendo el valor máximo un indicador de parecido entre las dos imágenes.

Aplicación práctica de Filtros con DTF

GRADIENTE Y DERIVADA

Entre los métodos de realzado de imágenes, se encuentra el del gradiente y derivada, el cual consiste en el realzado de los bordes de la misma, o lo que es lo mismo, realzar los cambios abruptos de la luminancia. Este filtro es el opuesto al anterior, el suavizado. Si el anterior filtro consiste en una integración de las luminancias de los pixels que rodean al pixel a tratar, es lógico suponer que una derivada o

Variando el valor central de la máscara, se consigue suavizar el suavizado o bruscalizarlo

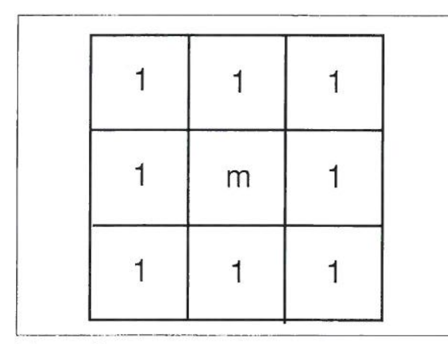


Figura 6.

diferenciación de los mismo resultará en el efecto opuesto, es decir, el resaltado de los bordes.

Uno de los sistemas más indicados para esta labor es la aplicación de un gradiente a la imagen. El gradiente (según la ecuación del gradiente de Roberts) es aproximadamente pro-

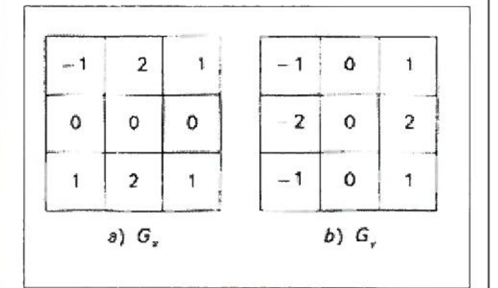


Figura 7.

porcional a la diferencia de luminancias entre los pixels adyacentes.

Se dividirá el calculo de gradiente por componentes, Gx y Gy, los cuales lograremos convolucionando la imagen con las máscaras de la figura 7.

Estás máscaras (denominados operadores Sobel) dan como resultado los gradientes en x e y para cada pixel. Componiendo ambos según la ecuación:

$$G[f(x,y)]=\sqrt{G_x^2+G_y^2}$$

Figura 8.

se obtiene la imagen gradiente de la imagen original.

Laplaciana

La máscara de la Laplaciana básicamente realiza la misma función que la del gradiente, solo que está más indicada para las imágenes en las cuales las variaciones de la intensidad a detectar son bastante más suaves. El problema principal de la Laplaciana, es su sensibilidad al ruido.

Existen sistemas para tratar de atenuar la sensibilidad de la Laplaciana al ruido, uno de ellos es la utilización de filtro ponderados con un pequeño filtro paso bajo. La implementación de esto, se puede lograr através de la aplicación de la IFT al perfil deseado en el dominio de la frecuencia. Esto nos devolverá una máscara bidimensional característica, o la PSF. Como ejemplo, se incluyen algunas máscaras que consiguen este efecto:

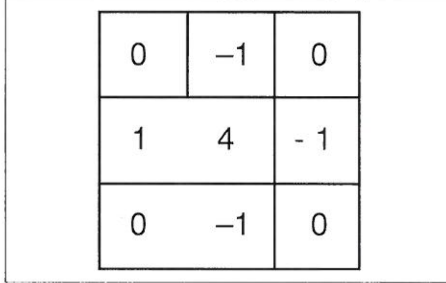


Figura 9.

Implementación en C

Con el software suministrado en el número anterior de esta revista venía una función de implementación de la DFT, esta viene adaptada para una imagen cuadrada de rango NxN. El código que se aporta en este número es una clase C++ para cargar TGA en MFC 3.0 (Sin retoque de paleta, es decir, funcionando con la paleta activa del CWnd activo) , además se incorpora código de TGA para DOS. La única diferencia radica en el tratamiento de la imagen como buffer lineal o como matriz cuadrada, fácilmente solventable si tenemos en cuenta que:

fila*columnas tot.+ columna parcial=posición en el buffer.

Para mayor claridad, supongamos una imagen 320*200, la fila 129 y columna 34:

129*200+34=25834

Si la imagen estuviese invertida, simplemente se complementa con el valor máximo del buffer. Esta ecuación, solventa el problema de implementación con la función DFT en el número anterior con una imagen cargada en un buffer lineal.

Esto nos devolverá una máscara bidimensional característica, o la PSF

BIBLIOGRAFÍA

Tratamiento digital de la imagen
(Alberto Domingo Ajenjo, Editorial Anaya)

Zen of graphics programimng
(Michel Abrash's, Coriolis Group)

Digital Halftoning
(Robert Ulichney, MIT Press)

Digital Image Warping
(George Wolberg, IEEE Computer society)

CONCLUSIÓN

El campo de aplicación de la DFT es enorme y juega un papel fundamental en el análisis y tratamiento digital de la imagen. La implementación de la DFT aunque costosa en tiempo es generosa en beneficios.

Existe un sistema, la transformada rápida de Fourier (FFT), menos costosa en tiempo, el cual será un tema a tratar con posterioridad en otro artículo.

UNA INTRO

Pedro Antón

Intros de 4K. Esta es la auténtica intro, el ejecutable no debe exceder de 4K, pero puede estar comprimido con los diversos compresores de ejecutables que existen en el mercado, los más empleados son el "LZEXE" de Fabrice Bellard y el "PKLITE" copyright de PKWARE, ambos reducen considerablemente el tamaño del ejecutable, siendo éste el que se debe ajustar a los 4096 bytes de tamaño máximo. En esta modalidad normalmente los programas no tienen música y el objetivo del programador es hacer la mayor cantidad de efectos de la forma más espectacular posible, para demostrar al resto de los competidores su superioridad.

Intros temáticas. El programador no tiene un límite de tamaño, pero dependiendo del objetivo del programa, ha de ajustarse al menor número de bytes posible. Esto quiere decir que si el programa se está realizando para dar publicidad a una BBS, se pretenderá incluir la Intro en todos los ficheros que distribuya dicha BBS, por lo tanto, es interesante que esta tenga el menor tamaño posible. Pero si la Intro se hace para anunciar la preparación de una demo o de una "party" no tiene ningún tipo de limitación, el programador empleará tantos bytes como crea necesarios para que el resultado sea lo suficientemente espectacular.

UNA INTRO TEMÁTICA.

Es necesario tener un tema, una canción y un efecto.

Este articulo, tratará la creación de una intro temática, es decir no se tendrá en cuenta el tamaño del programa, aunque se intentará realizar lo más reducido posible. Para llevar a cabo el desarrollo de un programa de estas características, son necesarias varias cosas:

Unas rutinas de sonido. En el ejemplo que acompaña al artículo, las rutinas han sido desensambladas del "player" de HSC creado por Chicken y ECR, para mayor comodidad de manejo. Se usa este formato ya que se trata de un reproductor de sonidos FM, a partir del chip OPL2, el cual es incluido en las tarjetas Adlib. Las tarjetas Sound Blaster y compatibles llevan el chip OPL3 o incluso el OPL4, y al igual que ocurre con los microprocesadores son compatibles con sus hermanos pequeños.

Una canción. La canción que acompaña a este artículo ha sido creada por David Tolosana, músico de iGUANA, en formato HSC.

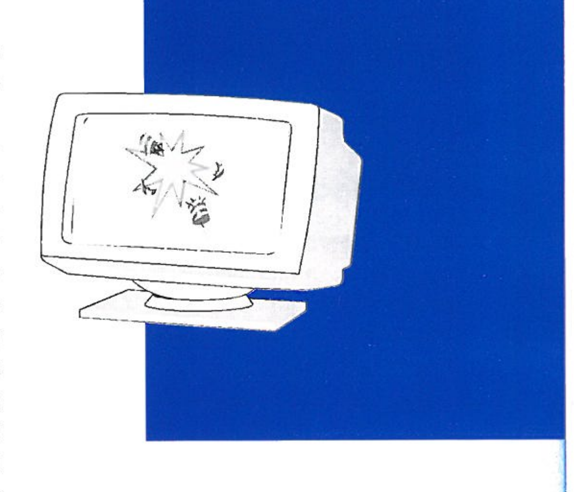
Un tema. El tema que se va a tratar en la intro es el curso que se está realizando en la revista, aderezada con una gran lista de saludos y agradecimientos.

Un efecto. En una intro se pueden incluir varios efectos, pero es evidente que cuantos más efectos tenga la intro, más código habrá que crear y por supuesto mayor tamaño tendrá el ejecutable. En el artículo del mes de Septiembre se pudo observar como hacer una intro con el efecto del plasma. En este artículo se explicará como realizar el efecto de la nieve.

LA NIEVE

Un efecto muy sencillo y vistoso.

Este efecto es realmente sencillo de programar y el resultado es bastante espectacular, no obstante es interesante añadir algo más al efecto para que la intro quede más espectacular, en este



Una Intro es un programa del menor tamaño posible, que generalmente incluye alguna melodía y contiene gran cantidad de mensajes junto con algún efecto o mejor aún con varios.

LA UNIDAD DEMOVGA

Es muy útil crear una librería con las funciones más empleadas en la elaboración de un trabajo. En el caso de este curso se creará una unidad que sea útil a la hora de confeccionar un efecto "demoero", y que se irá ampliando con cada artículo. Hasta el momento la librería posee los procedimientos mostrados en la figura.

Es muy útil crear una librería con las funciones más empleadas en la elaboración de un trabajo

A continuación se comentará brevemente la función de cada procedimiento. Los procedimientos McgaOn y McgaOff como su propio nombre indica, activan y desactivan el modo de vídeo MCGA. El procedimiento VerticalRetrace, también es de fácil interpretación, espera a que finalice el retrazo vertical. Los procedimientos PutColor, GetColor ponen o leen el número de color dado y CopyColor copia un color a otro. El procedimiento RotaPal, rota la paleta entre dos colores dados. El procedimiento PutPalette, pone una paleta completa indicándole como parámetros de entrada la dirección donde esta se encuentra almacenada, cabe destacar que han de estar definidos los 256 colores y los bytes deben ser, byte de "Red", byte de "Green" y byte de "Blue", como todos sabéis. PutPixel, DrawLineH y DrawLineV, pone un pixel, dibuja una línea horizontal y una vertical, en el segmento especificado.

El lector notará en el código la similitud entre el nombre del procedimiento y su función, a esta notación se le denomina notación húngara, y quizás se explique algo sobre ella en posteriores artículos.

Todos estos procedimientos, únicamente son perfectamente funcionales en modo MCGA, es decir, 320x200 en 256 colores.

LA UNIDAD OPLLIB

Esta unidad provee de las principales funciones para poder tocar canciones en cualquier tarjeta de sonido que incorpore un chip OPL. La unidad permite realizar las funciones mostradas en la figura.

La función de cada procedimiento es la siguiente. PlayNewSong toca una canción dado el segmento y el offset donde se encuentra almacenada ésta. StopMusic termina de tocar la canción que se encuentre sonando en ese momento. FadeOut, baja el volumen de la canción hasta hacerla inaudible. DetectOPL2 detecta si el hardware sobre el que se encuentra corriendo el programa posee un dispositivo con el chip OPL2 o superior. FillOPL2Vars rellena las variables declaradas públicas para posibles efectos con los datos que se encuentren sonando en ese instante.

Para más información sobre esta librería el lector puede leer el fichero CONTENTS.DOC que se adjunta en el fichero HSC_SRC.ZIP.

EL BUCLE PRINCIPAL

Unir todo esto da lugar a una bonita intro.

Como se ha dicho anteriormente en primer lugar se debe calcular los parámetros aleatorios de cada copo usado en la nevada. En segundo lugar se reservará memoria para tener una página virtual, y se limpia, a su vez se actualizarán las variables globales que van a controlar el programa. Al requerir 64000 bytes no es necesario usar la función MemAllocSeg, pero esta nos asegura un offset de cero en la memoria solicitada. Es recomendable guardar el segmento donde se ha localizado ese buffer. En tercer lugar se puede generar los colores que usará el programa. Acto seguido se procede a dibujar la ciudad, esta es dibujada en el buffer y en pantalla al mismo tiempo, de lo contrario, la ciudad se iría dibujando o borrando a medida que la nevada avanza. Como la rutina que limpia el buffer no lo hace por completo, los copos nunca borrarán la ciudad. El quinto paso, podría ser comenzar a tocar la canción, que previamente ha sido incluida en forma de procedimiento "external". El sexto paso será ya controlar la nevada y finalizar la intro al pulsar una tecla.

Para controlar la nevada se seguirán siempre los mismos pasos. Leer el valor de equalizado de cada canal, para que las luces de la ciudad parpadeen al ritmo de la música, modificando el color de cada luz roja de la ciudad. Realizar todos los pasos explicados para generar la nevada. Esperar un retrazo vertical antes de hacer el paso anterior. Controlar el número de frames que se han producido para así realizar un cambio en los textos. En caso de realizar un cambio se controla si hay que poner textos o si por el contrario hay que borrarlos. Notese que aunque el objetivo de este curso es "enseñar al que no sabe" para hacer cosas tan elementales como si el contador de textos es par o impar se recurre al código maquina, para ir familiarizando al lector con la mezcla de código de bajo nivel, con el de alto nivel, consiguiendo así una mayor velocidad. Actualizar la página virtual, bien limpiándola o bien, escribiendo en ella, los textos característicos. Repetir el proceso hasta que se produzca una pulsación de teclado.

En la "DemoScene" no necesariamente es mejor lo técnicamente muy complicado

El efecto es sencillo, pero aquí queda patente, que en la "demoScene" no necesariamente es mejor, lo técnicamente muy complicado. Con un efecto técnicamente sencillo se puede conseguir un espectacular resultado. Lo importante de este artículo es que el lector intente unir, textos, efecto(s) y sonido para crear algo sólido.



MEJORANDO LA CALIDAD DE SONIDO

Héctor Martínez

Hasta el momento, podemos reproducir cualquier tipo de fichero MOD, tanto en estéreo, como en 16 bits (siempre que el hardware lo permita). También se soporta la tarjeta Gravis UltraSound, con la que se consigue una calidad de sonido mucho mayor. Esto es debido a que interpola muestras por hardware.

Como apoyo a este artículo, sería conveniente tener a mano la tercera entrega de esta serie dedicada a la programación de sonido (número 6 de la revista). Ya que se verá como incrementar la calidad de sonido del MOD player. Para conseguir esto, tendremos que sacrificar el consumo de CPU, que comenzará a ser crítico. Para empezar, se explicará el control de saturación, ya que es muy sencillo. Posteriormente procederemos con la interpolación, que es lo que más incrementa la calidad del sonido y, a la vez, lo que más CPU consume.

¿QUÉ ES LA SATURACIÓN?

La saturación es un efecto que se produce cuando una señal ha llegado a los límites del canal por el que se transmite. Esto es lo que pasa, por ejemplo, con un aparato de música cuando subimos el volumen mas de lo que aguanta el aparato. Supongamos que nuestros altavoces soportan una amplitud máxima de 12 V. Puesto que el volumen del aparato de música, lo que hace es amplificar/atenuar la onda, todo ira perfecto mientras la amplitud de la onda que le llegue a los altavoces no supere los 12 V. Pero , ¿cuál será el efecto producido en el caso de que nos pasemos?. En este caso la respuesta es bastante sencilla, simplemente, la onda quedará recortada a esta amplitud, es decir, todos los valores por encima de 6V, se quedarán a 6V, y todos los que estén por de bajo de -6V, se quedarán a -6V. A este efecto se le llama saturación, y el sonido final que queda es muy característico. (ver figura 1).

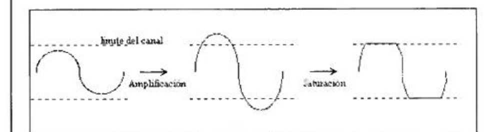


Figura 1.

PROBLEMA DE LA SATURACIÓN EN UN MOD PLAYER

Ante el problema de la saturación, no se puede hacer nada, simplemente bajar el volumen, pero el problema que tenemos nosotros en el MOD player es muchísimo más grave, ya que la distorsión aparece cuando comenzamos a sumar canales. En el caso del MOD, recordemos que el volumen máximo de un instrumento era 64. Si tenemos 4 canales, el nivel máximo al que llegaremos será 64*4=256 niveles, que es representable precisamente con 8 bits, pero, ¿qué pasaría en el caso de tener 8 canales? Lo que está claro es que si todos los canales tienen un instrumento sonando al máximo volumen, llegará un momento en el que al sumar un canal a lo que llevábamos acumulado, se generará un acarreo y el resultado de esa suma será falso (120+10=2!!!!). El efecto producido en la onda en este caso es el que se muestra en la figura 2

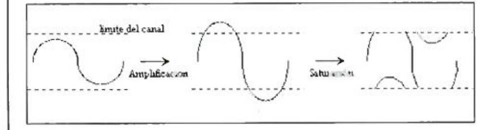
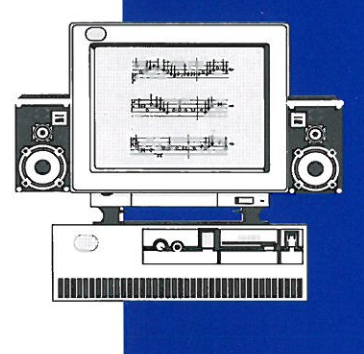


Figura 2.

En la figura no se ve el caso real por el que pasa esto en un MOD, ya que la satu-



En este artículo veremos como podemos conseguir la calidad de sonido de la GUS con tarjetas que no son Wave Table. Para ello deberemos implementar la interpolación mediante el software.

ración se produce al sumar dos canales, no al amplificar, pero el ejemplo es mucho mas claro viéndolo de esta manera.

En este caso el resultado final sí que es desagradable para el oído, porque suena como un crujido. Este efecto lo podemos arreglar mediante software; con ello conseguiremos simular la saturación que se consigue con un aparato de música; que aunque siga quedando la onda distorsionada, no es tan desagradable para el oído humano.

La manera de arreglarlo, es bien sencilla, consiste en detectar este acarreo después de la suma, y en caso de que exista, dejar el valor a 127 o -128 según si nos salimos por arriba o por abajo.

Los cambios realizados en la rutina de llenado de buffer del player que teníamos hasta el momento son los siguientes:

```
ADD [SI],AX  {Coloco el resultado en el buffer}
     {Control de saturación}
JNO @Sigue  {Si no hay overflow, no hay saturación, por lo tanto nos la saltamos}
MOV AX,-32768  {Si se ha salido por abajo, pondremos el valor mínimo}
JNS @PorDebajo
{Si el resultado es positivo, entonces es que antes del overflow era negativo, por lo tanto nos habremos salido por abajo}
     MOV AX,32767          {Si nos salimos por arriba, ponemos en AX el valor máximo}
     @PorDebajo:
MOV [SI],AX  {Metemos el resultado en el buffer de salida}
@Sigue
```

{A partir de este momento, las siguientes operaciones sobre esta muestra deberían dejar como resultado siempre el valor de la saturación (excepto si se llegara a contrarrestar las siguientes muestras), pero como se puede observar, si hemos dejado como valor el -32768 y en la próxima vuelta sumamos, el resultado no será correcto, ya que dejará de estar saturado cuando en realidad debería seguir estándolo. De todas maneras el oído humano no detecta tanto esto como el hecho de no controlar la saturación, y ante lo costoso que sería realizar esto último comentado, es mejor dejarlo así}

Con el control de saturación podemos incrementar el volumen más de lo teórico, de manera que se utilizan más bits por canal para cada instrumento y, por lo tanto, una calidad de sonido mayor. Tampoco conviene subir el volumen a un nivel en el que la saturación sea constante.

INTRODUCCIÓN A LA INTERPOLACIÓN

La interpolación se utiliza para mejorar notablemente la calidad de sonido de las digitalizaciones realizadas a baja frecuencia.

En un MOD, las digitalizaciones se reproducen después a unas frecuencias diferentes a la frecuencia que se han digitalizado según la nota que se está tocando. Para que el reproductor consiga la máxima calidad de sonido se muestrea a 44,1 KHz. Si la frecuencia de un cierto instrumento ha de ser 10 KHz, y nuestro player trabaja a 44,1 KHz, la única manera de conseguir que ese instrumento suene a la frecuencia deseada es repitiendo muestras, en este ejemplo repetiría unas 4 veces cada muestra (44,1 / 10). Esto provoca una señal de salida como la de la figura 3

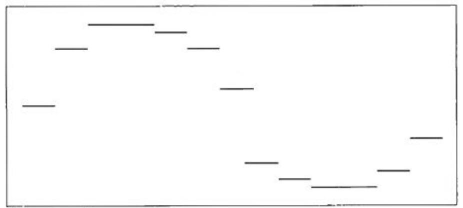


Figura 3.

Este tipo de onda, tiene un sonido tipo "lata". Para mejorar esto, lo que podemos hacer es lo que se llama interpolación lineal, que consiste en interpolar muestras entre un valor y el siguiente en lugar de repetirlas. Una vez aplicada la interpolación lineal, conseguimos una onda como la de la figura 4.

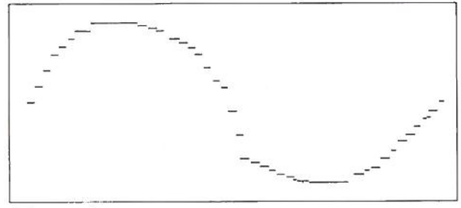


Figura 4.

Como se puede observar, se parece mucho mas a la señal original que antes de interpolar, y el sonido final conseguido es mucho mas limpio. En resumen, con la interpolación lineal, se consigue que suenen mejor incluso los sonidos digitalizados a baja frecuencia.

LA INTERPOLACIÓN APLICADA AL REPRODUCTOR DE MODS

Ahora pasaremos a ver como aplicar la interpolación lineal a nuestro player. En primer lugar, cabe remarcar que hay otros tipos de interpolaciones más complejos que la lineal, y que se aproximan mucho más a la señal original, pero a la hora de tener que implementarla por software, la más rápida para una CPU es la lineal, ya que sólo consiste en calcular las muestras intermedias entre un valor y el siguiente.

Aplicaremos la interpolación a cada canal para mejorar el sonido individual de cada uno antes de proceder a la mezcla. Para ello, aprovecharemos que hemos calculado el incremento en aritmética de punto fijo, en el cual el byte alto nos indicaba el valor absoluto a sumar, y el byte bajo, una especie de decimales, que en el caso de provocar acarreo, implicaba incrementar en uno la parte entera. Estos "decimales" los aprovecharemos para ponderar un valor entre una muestra y la siguiente. Si los decimales están cerca de 0, cogeremos un valor muy cercano a la muestra actual, y si los decimales se acercan a 255, cogeremos un valor cercano a la muestra siguiente. Como después de la última muestra del instrumento lo que hay en memoria es aleatorio, habrá que modificar los instrumentos para que en el caso de que haya loop la siguiente muestra a la última coincida con la primera muestra del loop, y para que la siguiente muestra a la última de un instrumento sin loop sea igual a la última ó a 0.

Ahora procederemos al algoritmo de la interpolación propiamente dicho. En primer lugar cogeremos la muestra siguiente, y la actual. Estos dos valores hay que restarlos, de manera que obtendremos la distancia entre dos muestras. Ahora, sólo hay que coger un valor intermedio entre esas dos muestras según el valor de los decimales, para ello, lo que se hace es multiplicar los decimales por la diferencia entre las dos muestras, y posteriormente dividir entre 256, que es el valor máximo de los decimales; de manera que si los decimales eran 0, el

resultado será 0, y si eran 255, el resultado será de *255/256 * diferencia*, es decir, prácticamente la diferencia completa. Este valor sumado a la muestra actual, nos indicará lo que tenemos que meter en el buffer final de salida. La manera de realizar esto en ensamblador es multiplicar, y después quedarnos con el byte alto del resultado. Como vemos, esto es muy costoso, ya que requiere de una multiplicación por cada muestra que tengamos que escribir en el buffer de salida.

EL PROGRAMA DE EJEMPLO

Pasaremos a comentar ahora los cambios realizados en el MOD player que teníamos hasta ahora, ya que no hace estrictamente lo que he explicado en las líneas anteriores.

{Interpolación lineal por software}

MOV AL,[ES:BX+1] {Leo muestra siguiente del instrumento}

SAR AL,1

{Divido la muestra entre dos, ya que de lo contrario, la resta entre dos muestras que oscilan entre -128 y 127, abarca los valores de -255 a 255, y esto no se podría codificar en un byte. Realmente, con esto perdemos precisión, pero si no lo hacemos así, necesitaríamos más registros e implicaría hacer PUSH y POP, cosa nada conveniente en este bucle que requiere gran velocidad.}

MOV AH,[ES:BX] {Se coge la muestra actual}

SAR AH,1

{Por el motivo explicado anteriormente, se divide entre dos, de esta forma los valores a restar están entre -64 y 63, de manera que la resta estará en un rango entre -127 y 127.}

SUB AL,AH {Se hace la resta tal y como se explicó}

TEST AL,$80 {Si el bit 7 es 1, quiere decir que es negativo}

JNZ @Negativo

{Este caso especial tampoco haría falta hacerlo, ya que bastaría con un IMUL, el problema es que si se utiliza IMUL, se consideran los dos operandos con signo, y el valor de los decimales no lo es, la manera de arreglarlo sería haciendo IMUL con registros de 16 bits, pero al igual que antes, no nos queda ninguno libre. Esto nos implica hacer un salto, nada agradable si lo que buscamos es velocidad}

{Caso positivo:}

MUL DH {Se multiplica la parte fraccionaria por la diferencia: AX=CL*AL}

SAL AH,1 {Se multiplica por 2 para contrarrestar la división por dos de antes}

MOV AL,[ES:BX] {Cogemos la muestra actual}

ADD AL,AH {y se le suma el valor calculado a la muestra}

JMP @Positivo {Se acaba la interpolación, en AL tenemos el valor a utilizar}

@Negativo:

NEG AL {En el caso de ser negativo, trabajaremos con su correspondiente positivo}

MUL DH {El resto es igual que en el caso positivo}

SAL AH,1

MOV AL,[ES:BX]

SUB AL,AH

{Se suma el valor calculado a la muestra. Como AH es negativo, Se hace mediante una resta}

JMP @Positivo

{Fin de interpolación lineal; en AL está la muestra final}

En el programa ejemplo, podremos notar la diferencia entre interpolar y no interpolar. Pulsando la tecla 'i' cambiaremos entre un modo y otro. A veces, con interpolación se oirán unos pequeños clicks, esto es debido a que al interpolar, el sonido es tan puro, que se llegan a oír los saltos provocados al cambiar de un instrumento a otro. Arreglar esto es bastante mas complicado. La tarjeta GUS, hace interpolación por hardware, pero en cambio los clicks los tenemos que eliminar mediante el software. En la figura 5 se puede ver un motivo por el que se puede producir un click.

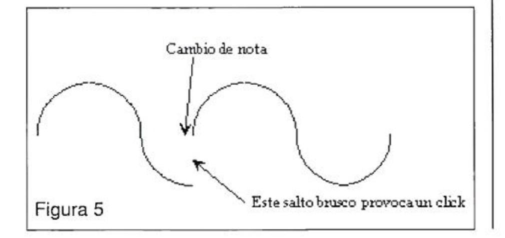


Figura 5

PRÓXIMAS ENTREGAS

El reproductor musical ya está muy completo, ahora sólo falta que implemente algunos efectos más, que tenga en cuenta el volumen independiente de cada canal, y que soporte algún otro tipo de formato de los más frecuentemente utilizados.

BUSCAMOS A LOS MEJORES

- Programadores en C/C++
- Grafistas, diseñadores, dibujantes
- Expertos en lenguaje ensamblador
- Programadores 3D
- Expertos en Sonido
- Músicos con nociones de MOD, MIDI
- Programadores de juegos
- Animadores gráficos
- Infografistas con experiencia en 3D Studio, Photoshop, Deluxe Paint Animation
- Programadores con dominio de lenguajes multimedia: Authorware, Toolbook, Visual Basic, Macromedia Director, etc
- Expertos en comunicaciones

PARA DESARROLLAR

SOFTWARE MULTIMEDIA:

Presentaciones, libros interactivos, programas educativos, sistemas de comunicaciones, centros servidores de datos, videojuegos y mucho más...

SI ERES programador, músico o infografista y tienes ideas o proyectos en estudio de desarrollo ponte ya en contacto con nosotros.

TE OFRECEMOS las mejores condiciones y apoyo para producir tus programas y venderlos en el mercado nacional y extranjero. Formación en nuevas tecnologías y la mejor biblioteca de rutinas gráficas y sonido para desarrolladores.

Aportamos gráficos, rutinas o música, según las necesidades, para complementar cada proyecto.

Si estás interesado en unirte a una de las empresas más punteras en alta tecnología y desarrollo de software, no dejes pasar esta oportunidad. Envíanos carta con tus datos personales (curriculum vitae, con una muestra de tus anteriores trabajos) y un teléfono de contacto a:



DIGITAL DREAMS MULTIMEDIA
Ref. Programadores
C/ Vicente Muzas 15, 1º D - 28043 MADRID
Tf.: (91) 519.23.53 Fax (91) 413.55.77
BBS: (91) 519.75.75 Internet: ddm@servicom.es

RAY TRACING (III) LA REFLEXIÓN

Álvaro Silgado

La reflexión de la luz es un fenómeno que se produce en la mayoría de las superficies. Añadiendo esta capacidad al programa, se abre un nuevo abanico de posibilidades visuales.

La iluminación, tal y como se estudió en el anterior capítulo, hace posible la generación de escenarios de cierta calidad. Se puede jugar con las luces y las sombras, obteniendo sensaciones muy reales de profundidad en lo que ya es un mundo tridimensional.

Los objetos que se diseñen pueden tener diferentes colores y tonalidades, pero todos ellos parecerán objetos "de goma". Esto es debido al tratamiento que se ha hecho de la luz. Toda la luz que incide en los objetos es absorbida en su totalidad por ellos, transmitiendo la luz correspondiente a su color.

En una escena real, la luz no sólo es absorbida por los objetos. También puede ser reflejada o refractada por ellos en mayor o menor proporción. Por ejemplo, ¿de qué color es un espejo? Esta pregunta puede parecer absurda, pero no es así. Un objeto es un material "sin color", o dicho de otra forma, es un material que no transmite luz, sino que la refleja en su totalidad.

Piense ahora en una bola de billar de color azul. ¿Qué es lo que se verá si se coloca sobre un tapete de color rojo? Parte de ella se verá azul y parte de ella, la que refleja el tapete, de color morado. Lo que se ha producido es una mezcla de colores en el ojo que capta la escena. Por un lado, se ve el azul original de la bola. Por otro lado, el rojo reflejado en ella. La suma de ambos colores es el morado obtenido.

En este capítulo se va a implementar esta cualidad de la luz, que permitirá obtener resultados sorprendentes. También se estudiará una técnica de aceleración conocida como "Bounding Boxes".

REFLEXIONES SOBRE LA REFLEXIÓN

Para poder reproducir el fenómeno de la reflexión de la luz, se puede echar mano de los apuntes de física del colegio, porque basta con recordar un par de leyes:

- Al incidir un haz de luz contra una superficie, el rayo incidente y el reflejado forman un mismo ángulo respecto al vector normal de dicha superficie en el punto de intersección. En la figura 1 se puede apreciar el vector normal a la superficie (N), el rayo incidente (I) y el rayo reflejado (R). El ángulo que forma el rayo incidente con el vector normal se llama ángulo de incidencia (Ai) y el que forma el rayo reflejado con el vector normal se llama ángulo de reflexión (Ar). Según esta ley física,

$$Ai = Ar$$

- El rayo incidente, el vector normal y el rayo reflejado están situados en un mismo plano, es decir, el vector reflejado se puede expresar como una combinación lineal de los vectores incidente y normal:

$$R = a * I + b * N$$

De lo que se trata es de calcular el rayo reflejado R. Para ello, hay que tener en cuenta que todos los vectores están normalizados (su longitud o normal vale 1). De esta forma, se puede ver claramente que

$$\cos(Ai) = -(N \mid I)$$
$$\cos(Ar) = N \mid R$$

Hay que recordar que el símbolo "|" representa el producto escalar de dos

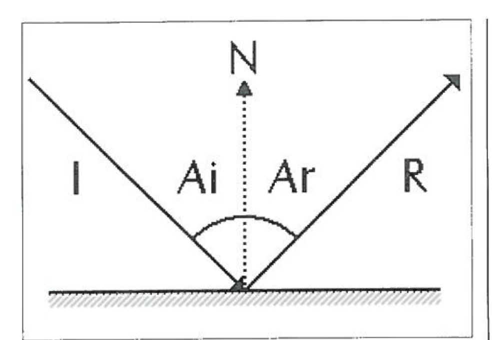


Figura 1. El ángulo incidente y el reflejado son iguales.

vectores, cuyo valor es el coseno del ángulo que forman dividido por el producto de sus normales. Como ambos vectores están normalizados, el resultado obtenido es el coseno del ángulo que forman entre sí.

Por otro lado, si los ángulos de incidencia y de reflexión son iguales, también lo son sus cosenos. Por lo tanto, se puede decir que

$$cos\ (Ai) = cos\ (Ar)$$

Desarrollando esta igualdad, se obtiene

$$-\ (N \mid I) = N \mid R$$

Sustituyendo la expresión de R obtenida anteriormente:

$$-\ (N \mid I) = N \mid (a * I + b * N)$$

Desarrollando este producto,

$$-\ (N \mid I) = a * (N \mid I) + b * (N \mid N)$$

Examinando el resultado obtenido, se puede ver que aparece el término N I N, es decir, el producto escalar del vector normal consigo mismo. Puesto que está normalizado, el resultado es 1, ya que N forma un ángulo de 0 grados consigo mismo, y el coseno de 0 es 1. De esta forma, se puede simplificar la expresión:

$$-\ (N \mid I) = a * (N \mid I) + b$$

Los términos "a" y "b" se han empleado para expresar que R es cualquier combinación de N y de I, es decir, que pertenece al mismo plano que éstos. Por lo tanto, se puede tomar un valor arbitrario para "a" y obtener el valor de "b" correspondiente. Por ejemplo, haciendo

$$a = 1$$
$$b = -2 * (N \mid I)$$

De esta forma, la expresión del vector reflejado en función del incidente y del normal a la superficie queda así:

$$R = I - 2 * (N \mid I)$$

LOS PRIMEROS REFLEJOS

Bien, ya es posible obtener un rayo reflejado de otro en un punto de intersección. Esto se ha implementado en la clase rayo, que ahora tiene un método nuevo llamado rayo_reflejado.

Lo siguiente que se va a necesitar es asignar a un material un coeficiente de reflexión, que servirá para saber qué porcentaje de la luz que incide sobre el objeto es absorbida y qué porcentaje reflejada. Este coeficiente se ha implementado como un nuevo campo llamado kr en la clase material, así que todos los constructores de los objetos han sido modificados para que inicien también este campo y otros que se explicarán más adelante.

Hasta ahora, el algoritmo para obtener el color de un pixel en la pantalla era muy simple: trazar el rayo que pasa por dicho pixel, analizar si hace intersección con algún objeto y, si lo hace, estudiar la luz en dicho objeto. Este algoritmo tiene que ser transformado un poco para que admita reflexiones, quedando de la siguiente manera:

- Trazar el rayo que pasa por el pixel a estudiar.
- Comprobar si hace intersección con algún objeto y, en caso afirmativo, quedarse con el que lo hace más cerca del origen del rayo.
- Estudiar la iluminación en el punto de intersección para obtener la luz difusa por el objeto.
- Si procede, calcular el rayo reflejado en el punto de intersección y trazar este nuevo rayo de forma recursiva, promediando el color del objeto con el color obtenido con el rayo reflejado. El color resultante será

*kr * luz_reflejada + (1 - kr) * luz_difusa*

REFLEJOS Y MÁS REFLEJOS

La última línea del algoritmo anterior empieza con la frase "Si procede ...". ¿Es que no siempre va a trazarse el rayo reflejado? La respuesta es no, y por varios motivos. Para empezar, si un objeto tiene un coeficiente de reflexión 0, es absurdo desperdiciar tiempo calculando el rayo reflejado, ya que al promediar se va a multiplicar por 0.

Pero aun en el caso de que kr sea mayor que 0, no siempre se trazará el

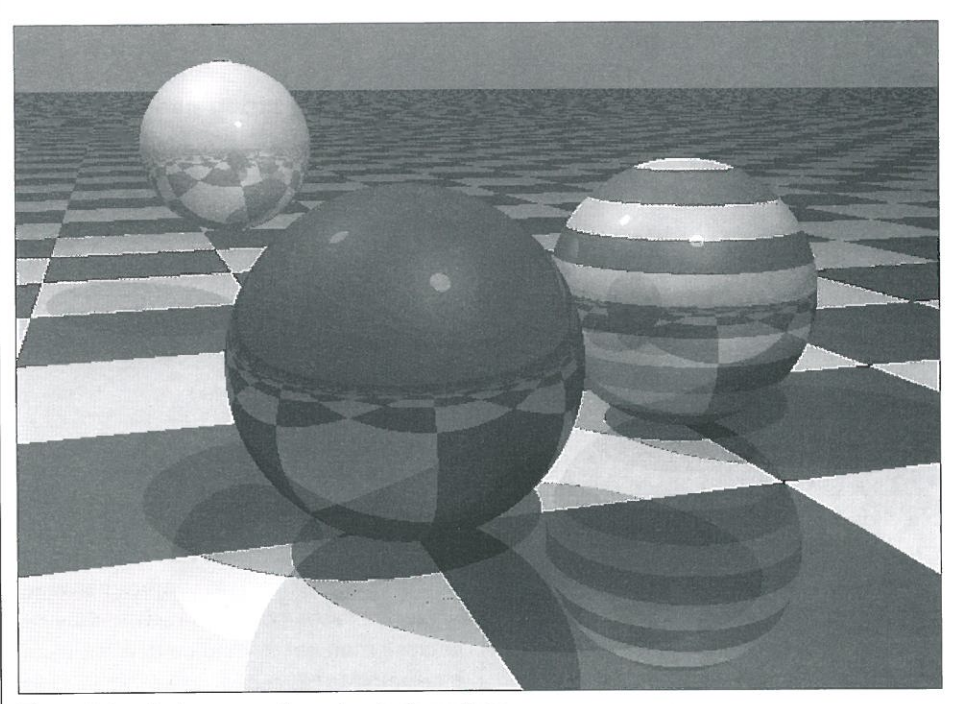


Figura 2. Las imágenes ya tienen bastante realismo.

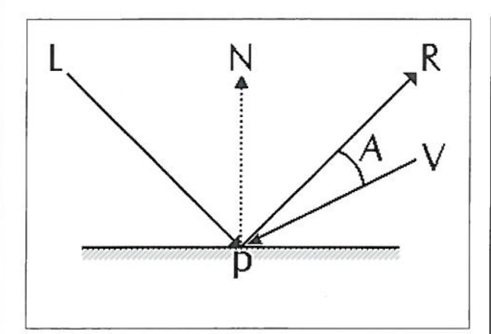


Figura 3. La luz especular depende de la posición del observador.

rayo reflejado, ya que el programa podría entrar en un bucle sin fin. Para entender esto no hay más que pensar en una habitación con al menos dos de sus paredes paralelas con un espejo en cada una a la misma altura. Si se mira en uno de los espejos lo que se ve es el reflejo de uno mismo, más el reflejo ocasionado por el espejo de atrás, más el reflejo del reflejo del reflejo... y así infinitas veces.

Para evitar que esto ocurra, se limita el número de veces que un mismo rayo puede hacer reflexión. Esto se hace con una variable que se le pasa al procedimiento *trazar_rayo*, que se va incrementando cada vez que se genera un rayo reflejado. Antes de generar un nuevo rayo, se comprueba si este valor es inferior al definido como máximo, y si no es así, no se genera. Con un máximo de 2 reflexiones se alcanzan resultados bastante reales. La figura 2 muestra una imagen obtenida con esta técnica. Otra forma de limitar el número de rayos reflejados es con los coeficientes de reflexión de los objetos con los que van rebotando. Para entender esto, supóngase que se traza un rayo y hace intersección con un objeto con kr = 0.1. Se genera un rayo reflejado que hace intersección con un segundo objeto con kr = 0.05. ¿Se debe generar un segundo rayo reflejado? No merece la pena. Si el color que proviene en el tercer rayo es C, la aportación del tercer rayo en el cómputo total del color es $C * 0.1 * 0.05 = C * 0.005$. Es decir, despreciable. Por lo tanto, el procedimiento *trazar_rayo* recibirá otra variable más, que es el producto de todos los coeficientes de reflexión con los que ha ido topando el rayo. Si esta variable es inferior al mínimo predefinido, no se generará un nuevo rayo reflejado.

Las dos constantes predefinidas que realizan esta "poda" de rayos se encuentran en el módulo RT.HPP.

Por último, cabe destacar una última modificación. El vector normal de la superficie era calculado en la función *iluminar*. Pero como también es necesario para el cálculo del rayo reflejado, este vector es calculado antes y pasado a dicha función.

LUCES ESPECULARES ESPECTACUALARES

El fenómeno de la reflexión no sólo repercute en que los objetos se reflejen unos en otros. También afecta a la iluminación de éstos. Si, por ejemplo, se apunta a un espejo con una linterna y un observador se coloca en la línea de reflexión de su haz de luz, no sólo verá la linterna reflejada, sino que además será iluminado por ésta. Esta iluminación se conoce con el nombre de luz especular, y contribuye a dar gran realismo a la escena, creando los típicos puntos luminosos en los objetos pulidos.

Hasta ahora se ha estudiado un tipo de iluminación conocida como luz difusa. Esta iluminación es la que reciben los objetos según su posición con respecto a las luces que iluminan la escena. Da igual el punto desde donde se contemple ésta: los objetos tienen siempre la misma iluminación. Por el contrario, la luz especular depende de la posición del observador, ya que es un reflejo directo de la luz que rebota en los objetos e incide sobre éste.

Como se puede observar en la figura 3, la cantidad de luz recibida por el observador depende del ángulo (A) formado por el vector de luz reflejado (R) y la dirección de visualización (V). Si "I" es el valor de la luz que incide contra el objeto, se puede decir que la contribución de luz especular en el punto P es:

$$Is = I * ks * (R \mid V)$$

Ks es el coeficiente de luz especular, e indica la cantidad de esta luz que el objeto irradia. Hay otro factor que influye en esta iluminación, y es el denominado potencial especular (ps), ambos han sido implementados como un campo más de la clase material. A medida que aumenta este valor, la luz especular se vuelve más intensa y concentrada en el punto de máxima iluminación. Si disminuye el potencial, se va haciendo más difusa. Los objetos metálicos, por ejemplo, tienen un potencial especular muy elevado (más de 50). Aplicando este potencial en la fórmula anterior, se obtiene:

$$Is = I * ks * (R \mid V) \wedge ps$$

Como se puede observar, para obtener este valor es necesario calcular el rayo reflejado de luz en el punto P, lo cual es costoso en tiempo de cálculo. Para evitar esto, se suele hacer una aproximación a este valor que apenas es distinguible visualmente. Esta aproximación consiste en equiparar el ángulo formado por "R" y "V" con el formado por "N" y "H", siendo éste último la suma del vector de luz (L) y el vector de visualización (V). Es decir,

$$H = L + V \text{ (normalizado)}$$

De esta forma, se puede decir que

$$R \mid V = N \mid H \text{ (aprox.)}$$

Aplicando esta aproximación, la expresión de la iluminación especular queda así:

$$Is = I * ks * (N \mid H) \wedge ps$$

Si ahora se añade la expresión que se ha usado hasta ahora, se obtiene:

$$I = A + Ii * [\, P * (L \mid N) + ks * (N \mid H) \wedge ps \,] / Di$$

con i = 1 .. num_luces

"Ii" es la luz i-ésima que incide sobre el objeto en el punto P. Descomponiendo esta expresión en sus componentes roja, verde y azul ("r", "g" y "b" respectivamente),

$$r = Ar + Iir * [\, Pr * (L \mid N) + ks * (N \mid H) \wedge ps \,] / Di$$
$$g = Ag + Iig * [\, Pg * (L \mid N) + ks * (N \mid H) \wedge ps \,] / Di$$
$$b = Ab + Iib * [\, Pb * (L \mid N) + ks * (N \mid H) \wedge ps \,] / Di$$

con i = 1 .. num_luces

Para acelerar un poco los cálculos, se ha escrito una rutina que calcula de



forma rápida un número real elevado a un entero positivo. Otra optimización que se ha implementado es comprobar si llega algo de luz al objeto antes de realizar los cálculos, ya que como va disminuyendo en función de la distancia, llega un momento en el que queda completamente atenuada.

"BOUNDING BOXES": LAS CAJAS ACELERADORAS

A medida que el lector vaya introduciendo más objetos en sus escenas y añada reflexiones, los tiempos de generación de las imágenes serán más y más altos. Para intentar paliar un poco este descenso de velocidad, se ha implementado lo que se conoce con el nombre de "Bounding Box" o "Caja que encierra" (BBox a partir de ahora).

La idea es obtener las coordenadas mínimas y máximas de cada objeto que se incluya en escena. De esta forma, si un rayo no hace intersección con un BBox, tampoco lo hará con el objeto que

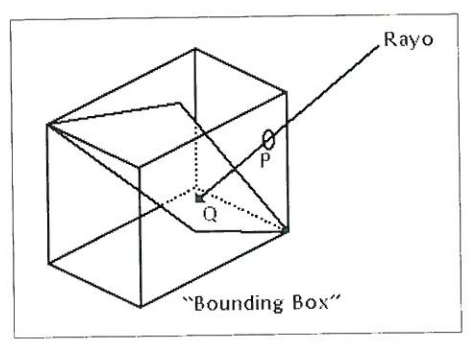


Figura 4. Primero se comprueba la intersección con el "Bounding Box" del objeto.

está en su interior. Así, es posible evitar hacer intersecciones complicadas de ciertos objetos. Aunque otras veces, puede resultar una pérdida de tiempo, ya que al final se termina haciendo dos intersecciones por cada objeto: una con su BBox y otra con el objeto. En la figura 4 se puede apreciar este caso. En general, resultará una técnica efectiva si se utilizan objetos con intersecciones complicadas (por ejemplo, polígonos).

Para poder desarrollar esta técnica, será necesario implementar un método para construir el BBox para cada objeto. Así, el BBox de una esfera se obtiene sumando el radio a las coordenadas del centro (el vértice superior de la caja) y restándolo (el vértice inferior). Para un polígono, será necesario recorrer todos su vértices y guardar las coordenadas mínimas y máximas. El suelo no tiene BBox y tampoco lo tiene un plano, ya que ambos son infinitos.

Lo interesante aquí es que un BBox representa una caja, con todos sus planos paralelos dos a dos, y todos ellos paralelos o perpendiculares a los ejes de coordenadas. De esta forma, para hacer la intersección con un BBox no hay que hacer la intersección con seis planos. Hay trucos que aceleran este proceso.

La primera idea a tener en cuenta es que desde cualquier punto sólo es posible ver tres caras de una caja como mucho. Por lo tanto, el rayo que se está estudiando sólo tiene tres planos candidatos para estudiar la intersección.

Seleccionar estos tres planos es fácil sabiendo de dónde proviene el rayo. Para ello, se comprueba para cada coordenada si es menor que la mínima del BBox, mayor que la máxima del BBox o intermedia. En los dos primeros casos, se puede obtener uno de los planos candidatos y se puede asegurar que el punto de origen del rayo está fuera del BBox (si el rayo parte de dentro del BBox, siempre hace intersección).

Para los planos candidatos obtenidos, se calcula la intersección con el rayo, seleccionando la que se produce más lejos del origen del rayo.

Una vez obtenido el punto de intersección, se comprueba si está dentro de los márgenes definidos por el BBox. Si es así, el rayo hace intersección.

PISA EL ACELERADOR...

Aplicando el concepto de BBox al programa, es posible reducir notablemente los tiempos de generación de imágenes. De esta forma, para comprobar si un rayo hace intersección con una esfera, primero se hará la intersección del rayo con su BBox. Si no hace intersección, tampoco lo hace con la esfera y no es necesario continuar. Si hace intersección, se calcula la de la esfera como antes.

En el caso del polígono, el tratamiento es diferente:

- Primero, se hallará la intersección con el plano en el que está incluido, ya que esta intersección es menos costosa en tiempo de cálculo que la del BBox.

- Luego, se comprobará si el punto está a una distancia menor que la del último punto obtenido. Para hacer esto, se han modificado los algoritmos de intersección de forma que siempre reciben la distancia a la que se ha producido la intersección más próxima hasta ese momento. Si está situado más lejos o a una distancia negativa, no es necesario continuar.

- A continuación, se comprueba si el punto está dentro del BBox, lo que como mucho costará seis comparaciones. No es necesario hacer la intersección del rayo con el BBox, puesto que ya se tiene el posible punto de intersección. Sólo hay que verificar si éste está dentro de los márgenes definidos por el BBox.

- Si las anteriores condiciones se han cumplido, se comprueba si el punto está dentro del contorno del polígono como se hacía antes.

Como se puede comprobar, no se ha ganado en tiempo para averiguar si un rayo hace intersección con un objeto. Pero sí se ha ganado en tiempo para saber si un rayo NO hace intersección con un objeto, lo cual es sutilmente diferente.

ÚLTIMAS CONSIDERACIONES

A medida que se vayan explicando los conceptos más técnicos sobre la técnica Ray Tracing, esta serie de artículos va a dejar de ser tan técnica para pasar a ser más práctica y creativa. Es de esperar que el lector disfrute con las nuevas imágenes que obtenga, mucho más reales que las anteriores gracias al fenómeno de reflexión de la luz. Queda invitado a enviarnos sus imágenes o animaciones preferidas que hayan sido obtenidas con este programa. De todas las recibidas se seleccionarán las mejores y se usarán para ilustrar esta serie de artículos o se publicarán en los CD-ROM de la revista.

BIBLIOGRAFÍA

- "An Introduction to Ray Tracing". Editado por Andrew S. Glassner. Editorial Academic Press.
- "Fractal programming and Ray Tracing with C++". Roger T. Stevens. Editorial M&T Publishing, Inc.
- "Graphics Gems". Editado por Andrew S. Glassner. Editorial Academic Press.
- "Fundamentals of Three-Dimensional Computer Graphics". Alan Watt. Editorial Addison-Wesley.

PROGRAMACIÓN DEL GDI (II)

Jorge R. Regidor

En este último artículo del curso de iniciación de programación para Windows, se va a tratar la segunda parte sobre la programación del GDI. Por ello, se pretende desarrollar el tema de la programación de las brochas, pinceles y bitmaps.

Al igual que un pintor dispone de sus herramientas de dibujo y pintura, en Windows los programadores y usuarios disponen de forma análoga de dichas herramientas. El GDI nos permite utilizar entre otras los pinceles, las brochas e incluso de utilizar imágenes ya creadas (bitmpas).

Existe para cada una de estas "herramientas" un conjunto de funciones, estructuras y macros, que nos permitirán de forma sencilla dar un aspecto más artístico a nuestras aplicaciones, sin por ello tener que ser un virtuoso del pincel.

Objetos GDI

Ya en el anterior artículo se introducía el concepto de objeto GDI, en el presente se va a explicar con mayor detalle la forma de utilizarlos, centrándonos en los pinceles, las brochas y los bitmaps. Todos tienen en común el proceso de utilización, que pasa por varias etapas como son:

* Creación del handle al objeto.
* Selección del objeto dentro del contexto de dispositivo.
* Guardar el objeto del contexto de dispositivo anterior.
* Utilización del objeto.
* Restaurar el objeto anterior.
* Destruir el objeto utilizado.

El proceso de creación del objeto utiliza funciones particulares para cada uno de estos, y dentro de cada objeto existen varias formas de crearlos.

La selección del objeto dentro del contexto de dispositivo, así como salvar el objeto anterior se realiza mediante la llamada a la función *SelectObject*, cuyo formato es el siguiente:

HGDIOBJ SelectObject(HDC hdc, HGDIOBJ hgdiobj);

donde,

hDC es el contexto de dispositivo donde se selecciona.

HGDIOBJ es el objeto a seleccionar. Este parámetro será del tipo HPEN, *HBRUSH, HFONT...* según el objeto a seleccionar.

La función devuelve el handle del objeto previo, lo que nos servirá para almacenar el objeto anterior.

La selección de un objeto dentro de un contexto de dispositivo, implica por ejemplo y en el caso de los pinceles, que cualquier línea que se dibuje, se hará con el ancho, color y estilo definidos para el pincel seleccionado.

Para la utilización del objeto, también existen diversas funciones, que permiten desde dibujar una simple línea a escalar una imagen, pasando por ejemplo, por rellenar el fondo de un rectángulo.

Para restaurar el objeto anterior se debe llamar de nuevo a la función *SelectObject*, pasándole ahora el handle del objeto devuelto por la llamada anterior.

El último paso, es quizá el más importante y pero también es el que más fácil se olvida uno de incluir, debido a que no se advierte de ninguna forma su no presencia hasta que se observa como los recursos de memoria del sistema descienden. Para eliminar un objeto después de su utilización, debe utilizarse la función *DeleteObject*, cuyo formato es el siguiente:

BOOL DeleteObject(HGDIOBJ hgdiobj);

donde,

hgdiobj es el handle del objeto a liberar. La función devuelve un valor distinto de cero si el proceso de liberación ha sido correcto, y cero si ha existido algún error.

PINCELES (PENS)

Los pinceles son objetos que el GDI utiliza para dibujar todas las líneas que se

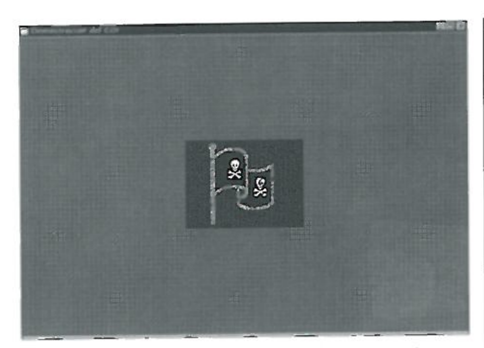


Figura 1. Aspecto del programa ejemplo.

necesiten en el programa, esto incluye no sólo las líneas sino además los bordes de los rectángulos, elipses, circunferencias, etc...

Las propiedades de los pinceles son el color, el estilo y el ancho. Para definir estas propiedades de un pincel existe una estructura llamada LOGPEN, cuyo formato es:

```
typedef struct tagLOGPEN
{
    UINT      lopnStyle;
    POINT     lopnWidth;
    COLORREF lopnColor;
} LOGPEN;
```

donde,

lopnStyle define el estilo del pincel, descrito en la Figura 1.

lopnWidth indica la anchura del pincel en las unidades lógicas definidades en el contexto de dispositivo.

lopnColor es un valor RGB que indica el color del pincel.

La funciones para crear un pincel son *CreatePen* y *CreatePenIndirect*, con los siguientes formatos:

*HPEN CreatePenIndirect(LOGPEN FAR * lplgpn);*

donde,

lplgpn es un puntero lejano a una estructuta LOGPEN que contiene la información necesaria para crear el pincel.

La otra opción es:

HPEN CreatePen(int fnPenStyle, int nWidth, COLORREF clrref);

donde,

fnPenStyle representa el estilo del pincel, descrito en la Figura 1.

nWidth es la anchura del pincel.

clrref es el color del pincel.

Las dos funciones devuelven el handle al objeto creado.

BROCHAS (BRUSHES)

Las brochas tienen una función muy especifica, como es rellenar superficies. Las propiedades que definen las brochas son el estilo, el color y el entramado. Estas propiedades quedan definidas en una estructura llamada *LOGBRUSH*, cuyo formato es el siguiente:

```
typedef struct tagLOGBRUSH
{
  UINT      lbStyle;
  COLORREF lbColor;
  int       lbHatch;
} LOGBRUSH;
```

donde,

lbStyle es el estilo de la brocha y puede tener los siguientes valores:

BS_DIBPATTERN: Indica que la brocha está definida por un bitmap independiente de dispositivo(DIB).

BS_HATCHED: Indica que la brocha es un entramado.

BS_HOLLOW: Sin brocha o transparente.

BS_PATTERN: Indica que la brocha está definida por un bitmap en memoria.

BS_NULL: Igual que BS_HOLLOW.

BS_SOLID: Indica que la brocha está formada por un sólo color.

lbColor es el color en el cual la brocha dibujará. Si el estilo es *BS_HOLLOW* o *BS_PATTERN*, este campo será ignorado. De ser el estilo *BS_DIBPATTERN*, la palabra de menor peso del campo debe indicar el modo en el cual se van a interpretar los índices de color del DIB. Los posibles valores para este campo son:

DIB_PAL_COLORS: Que indica que los índices de la tabla indexan a la paleta lógica de colores .

DIB_RGB_COLORS: Que indica que los índices del DIB indican valores RGB.

El último campo es lbHatch, e indica el modo de entramado en el caso de ser *BS_HATCHED*. Los posibles valores de entramado son:

HS_BDIAGONAL: Entramado a 45 grados ascendente. (De izquierda a derecha).

HS_CROSS: Entramado horizontal y vertical en cruz.

HS_DIAGCROSS: Entramado diagonal a 45 grados en cruz.

HS_FDIAGONAL: Entramado a 45 grados descendente.(De izquierda a derecha).

HS_HORIZONTAL: Entramado horizontal.

HS_VERTICAL: entramado vertical.

Si el estilo es *BS_PATTERN*, *lbHatch* debe contener el handle al bitmap que define el patrón.

Si el estilo es *BS_DIBPATTERN*, lbHatch debe contener el handle al DIB. Por último, si el estilo es *BS_SOLID* o *BS_HOLLOW*, este parámetro es ignorado.

FUNCIONES ASOCIADAS A BROCHAS

Existen varias funciones para crear y utilizar las brochas, entre las que se van a destacar algunas. La principal función para crear brochas es *CreateBrushIndirect*,con el siguiente formato.

*HBRUSH CreateBrushIndirect(LOGBRUSH FAR * lplb);*

donde,

lplb es un puntero a una estructura *LOGBRUSH* vista en el apartado anterior.

La función devuelve el handle a la brocha creada en memoria.

Y las otras funciones son:

CreateHatchBrush para crear brochas con entramado.

CreateSolidBrush para crear brochas de un color sólido.

CreatePatternBrush para crear brochas a partir de un bitmap.

Tanto las brochas como los pinceles deben de ser seleccionados dentro de un contexto de dispositivo para ser utilizados. Una vez seleccionados con SelectObject, cualquier acción que implique dibujar una línea será realizada con ese pincel, y cualquier acción que

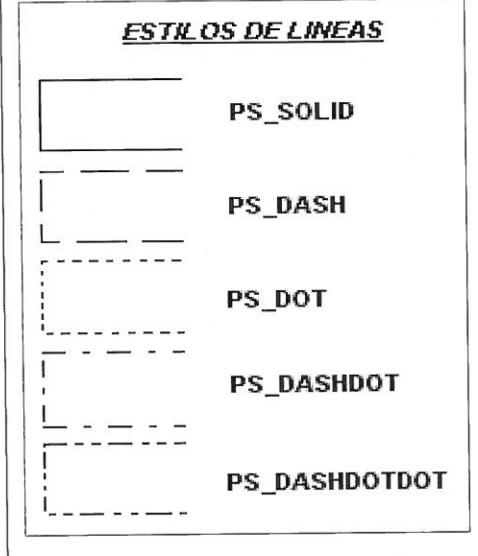


Figura 2. Estilos de los pinceles en Windows.

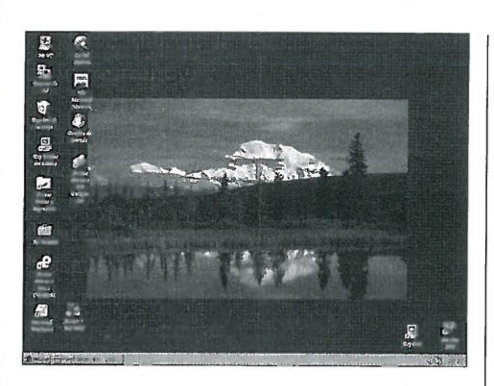


Figura 3. Fondo de Windows 95 con una brocha de patrón y un bitmap.

implique rellenar una superficie será utilizando dicha brocha.

MAPAS DE BITS (BITMAPS)

El formato estándar para manejo de gráficos dentro de Windows es sin ninguna duda el formato BMP. Este formato es dependiente del sistema, es decir, depende de los colores definidos en la paleta gráfica del sistema, por lo que la portabilidad de gráficos con este formato puede dar origen a cambios de color. Para solventar este problema, a partir de la versión 3.0 de Windows, se introdujo el formato DIB (Device Independent Bitmap), que extendía las capacidades del formato BMP, permitiendo ser independiente de la paleta de colores de cada equipo. Se ha realizado está introducción a modo indicativo, ya que este tema sale de los objetivos de este curso. Lo que si vamos a ver es una forma (quizá la más sencilla) que nos permita utilizar imágenes en nuestras aplicaciones, sin por ello, invertir demasiado tiempo en ello.

El método que vamos a exponer para añadir gráficos a nuestras aplicaciones usa los recursos de la aplicación, por lo que deberemos crear el bitmap desde el editor de recursos, o bien añadirlo a este. Una vez creado el bitmap sobre el fichero de recursos, deberemos cargarlo en memoria para poder utlizarlo. Para ello, se va a utilizar la función *LoadBitmap* cuyo formato es:

```
HBITMAP LoadBitmap(HINSTANCE hInst, LPCSTR lpszBitmap);
```

donde,

hInst es el handle a la instancia de la aplicación o librería donde se encuentra el bitmap a cargar.

lpszBitmap es el nombre dado al bitmap en los recursos. También se puede utilizar la macro *MAKEINTRESOURCE* para acceder al bitmap si ha sido definido mediante un identificador.

La función devuelve el handle al bitmap ya cargado en memoria.

¿Y QUÉ HACEMOS CON EL HANDLE AL BITMAP?

El método de plasmar un bitmap sobre nuestra ventana es un poco diferente al que se utiliza para los pinceles y las brochas. En primer lugar, debemos crear una copia en memoria del contexto de dispositivo de la ventana, operación que se realiza llamando a la función *CreateCompatibleDC*, este contexto de dispositivo en memoria servirá para trabajar sobre el bitmap en memoria y luego actualizarlo sobre el contexto real.

Existe una estructura que define las propiedades del bitmap y que una vez cargado éste debe ser llenada mediante la función GetObject, cuyo formato es:

```
int GetObject(HGDIOBJ hgdiobj,int cbBuffer,void * lpvObject)
```

donde,

hgdiobj es el handle del objeto sobre el que queremos obtener información (en nuestro caso un bitmap).

cbBuffer es el tamaño de la estructura a llenar.

lpvObject es un puntero a dicha estructura.

la función devuelve el número de bytes escritos si es correcto y cero si ha existido algún error.

En nuestro caso el formato de está función sería algo como:

```
GetObject(hBmp, sizeof(BITMAP), lpBitmap);
```

y se llenará una estructura del tipo BITMAP definida como:

```
typedef struct tagBITMAP
{
  int   bmType;
  int   bmWidth;
  int   bmHeight;
  int   bmWidthBytes;
  BYTE  bmPlanes;
  BYTE  bmBitsPixel;
  void FAR* bmBits;
} BITMAP;
```

donde,

bmType es el tipo de bitmap.

bmWidth es el ancho del bitmap en pixels.

bmHeight es el alto del bitmap en líneas raster.

bmWidthbytes indica el número de bytes por cada línea raster.

bmPlanes indica el número de planos de color del bitmap.

bmBitsPixel indica el número que bits que definen un sólo pixel.

bmBits es un puntero a los bytes que definen el bitmap.

Después de obtener la información referente al bitmap, se debe realizar el proceso para mostrarlo. Para ello debemos seleccionarlo dentro del contexto de dispositivo de memoria llamando a la función *SelectObject* y pasándole el handle a dicho contexto. Una vez realizado esto, ya podemos pintar el bitmap en pantalla, y para ello utilizaremos la función *BitBlt*, cuyo formato es:

```
BOOL BitBlt(hdcDest, nXDest, nYDest,
nWidth, nHeight, hdcSrc, nXSrc,
 nYSrc, dwRop);
```

donde,

HDC hdcDest es el handle al contexto de dispositivo donde queremos pintar el bitmap.

int nXDest es la coordenada X de la esquina superior izquierda donde se va a pintar el bitmap.

int nYDest es la coordenada Y de la esquina superior izquierda donde se va a pintar el bitmap.

int nWidth es el ancho del bitmap.

int nHeight es la altura del bitmap.

HDC hdcSrc es el handle de memoria donde está seleccionado el bitmap.

int nXSrc es la coordenada X de la esquina superior izquierda de donde se va a extraer el bitmap.

int nYSrc es la coordenada X de la esquina superior izquierda de donde se va a extraer el bitmap.

DWORD dwRop indica el modo de pintar el bitmap sobre el contexto de dispo-

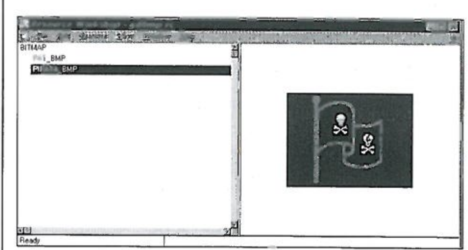


Figura 4. Bitmap desde el editor de recursos.



OPERACION RASTER	SIGNIFICADO
BLACKNESS	Pone el destino en negro
DSTINVERT	Invierte el bitmap destino
MERGECOPY	Combina el origen y destino con la operación AND
MERGEPAINT	Combina el bitmap origen invertido con el destino mediante la operacion OR
NOTSRCCOPY	Copia el bitmap origen invertido sobre el destino
NOTSRCERASE	Invierte el resultado de convinar con la operación OR el origen y el destino
PATCOPY	Copia el bitmap patrón sobre el destino
PATINVERT	Combina el bitmap patrón utilizando la operación XOR
PATPAINT	Combina el bitmap patrón invertido utilizando la operación OR
SRCAND	Combina los bitmaps utilizando el operador AND
SRCCOPY	Copia el bitmap origen sobre el destino
SRCERASE	Invierte el bitmap destino y lo combina con el origen utilizando la operación AND
SRCINVERT	Combina los bitmaps utilizando el operador XOR
SRCPAINT	Combina los bitmaps utilizando el operador OR
WHITENESS	Pone el destino en blanco

Tabla 1. Tipos de operaciones para combinar los bitmaps en la ventana destino.

sitivo. Ver la Tabla 1, donde se describen todos los posibles valores.

La función devuelve TRUE si la operación se realizó con éxito y FALSE en caso contrario.

Con esto, sólo queda eliminar los objetos de memoria mediante la función *DeleteObject*. Como se puede ver el método es sencillo y permite incluir imágenes a nuestra aplicación lo que con un poco de imaginación nos permitirá darle un aspecto mucho más profesional y agradable.

EL PROGRAMA DE EJEMPLO

El programa de ejemplo muestra como podemos cambiar el fondo de una ventana, añadiéndole un aspecto como el que puede tener el fondo en Windows. Para ello se ha creado una brocha con un bitmap patrón, cuyo tamaño debe de ser de 8x8 pixels. Además se pinta un bitmap en el centro de la pantalla, que se centra automáticamente. Por otra parte, moviendo el ratón y pulsando su botón derecho se dibujar líneas rectas para ilustrar la funcionalidad de los pinceles.

En cuanto a la parte de programación referente a este artículo, se ha enfocado todo sobre la función *Onpaint*, donde se crea la brocha mediante la función *CratePatterBrush*, se rellena el fondo con la función *PatBlt* y por último se pinta el bitmap con el método expuesto en el articulo. El otro punto a ver es el mensaje *WM_LBUTTONDOWN*, donde se pinta las líneas con el pincel, aunque como se puede observar es bastante sencillo de interpretar.

Sólo me queda decir que realicéis pruebas sobre este programa, cargando vuestros propios bitmaps, cambiando de tipo brochas y pinceles, etc...

CONCLUSIÓN

Espero que estos dos artículos sobre el GDI hayan servido para aclaran un poco esta parte tan importante de la programación para Windows, sobre la que recae el peso de toda la programación gráfica de la aplicación. Lo visto aquí no es más que la punta del iceberg, ya que existen muchísimas más funciones y objetos en el GDI, pero claro está que se escapan de los objetivos de este curso de introducción a la programación para Windows.

OPERACION RASTER	SIGNIFICADO
BLACKNESS	Pone el destino en negro.
DSTINVERT	Invierte el bitmap destino.
MERGECOPY	Combina el origen y destino con la operación AND.
MERGEPAINT	Combina el bitmap origen invertido con el destino mediante la operación OR.
NOTSRCCOPY	Copia el bitmap origen invertido sobre el destino.
NOTSRCERASE	Invierte el resultado de convinar con la operación OR el origen y el destino.
PATCOPY	Copia el bitmap patrón sobre el destino.
PATINVERT	Combina el bitmap patrón utilizando la operación XOR.
PATPAINT	Combina el bitmap patrón invertido utilizando la operación OR.
SRCAND	Combina los bitmaps utilizando el operador AND.
SRCCOPY	Copia el bitmap origen sobre el destino.
SRCERASE	Invierte el bitmap destino y lo combina con el origen utilizando la operación AND.
SRCINVERT	Combina los bitmaps utilizando el operador XOR.
SRCPAINT	Combina los bitmaps utilizando el operador OR.
WHITENESS	Pone el destino en blanco.

BIBLIOGRAFÍA

- Computer Networks & ISDN Systems, vol.27, no. 2, Nov.94
  Special Issue: Selected Papers of the First WWW Conference
- Computer Networks & ISDN Systems, vol.27, no. 6, Apr.95
  Proceedings of the Third International WWW Conference
- Tutorial Notes of the Third International WWW Conference
- IEEE Computer, vol.27, no.10, Oct.94
  Ronald J. Vetter, Chris Spell, Charles Ward
  Mosaic and the World-Wide Web, pag 49-57
- T. Berners Lee, "The HTTP Protocol as Implemented in W3"
  ftp://info.cern.ch:/pub/www/doc/http-spec.txt.Z
- "World-Wide Web Growth", ftp://nic.merit.edu

BASES DE DATOS: ADABAS

José María Peco

El pasado mes se inició bajo el titulo SISTEMAS GESTORES de BASES de DATOS una serie de artículos con los que se pretende explicar la mecánica seguida por los sistemas gestores para el mantenimiento y recuperación de la información de la Base de Datos.

De acuerdo con la clasificación expresada en el pasado artículo, el sistema gestor que se va a tratar este mes, pertenece al grupo de *Gestores Basados en Listas Invertidas*. Esta denominación es poco conocida, pero como se vera a continuación define perfectamente el sistema en el que se fundamenta la Base de datos a la que sirve.

ESTRUCTURA DE LA BASE DE DATOS:

La información contenida en la Base de datos se almacena físicamente en dos grandes ficheros, denominados *"DATA"* y *"ASSO"* si bien existe otro fichero, el *"WORK"*, que sólo es utilizado por el propio gestor para la realización de su misión de gestión.

FICHERO DATA:

Este fichero contiene realmente los datos de aplicación del usuario. La información contenida en él se encuentra agrupada en archivos lógicos, pudiendo tener hasta **255 archivos**, tal y como se muestra en la figura 1.

A su vez, cada uno de estos archivos lógicos, tiene la información agrupada en registros, pudiendo alcanzar el número de **16.777.216** registros cada fichero.

Por su parte cada registro puede tener hasta **500** campos, soportando los tipos que se describirán mas adelante.

Con el fin de agilizar las lecturas de disco, y, puesto que el sistema operativo lee y graba la información del disco por **bloques físicos**, el gestor agrupa en bloques los registros del mismo fichero, grabando y recuperando bloques completos de registros. Cada **bloque** se encuentra identificado por un número que determina su dirección relativa a principio del fichero. Es el *RABN* o Relative Address Bloque Number.

La figura 2 muestra de una forma esquemática la estructura de este fichero.

Para poder manejar toda esta información de una forma ágil, en la definición física del registro, se definen como índices determinados campos elementales (aquí denominados *DESCRIPTORES*) o bien agrupaciones de varios campos elementales o subcampos (denominados *SUPERDESCRIPTORES*).

FICHERO ASSO: ASSOCIATOR

Este fichero está íntimamente ligado al DATA, y su misión es la de permitir la recuperación de la información de una forma rápida, pues es el que contiene la tabla de conversión de direcciones y las listas invertidas que veremos a continuación.

Esta dividido en tantas partes como *archivos lógicos* contenga el DATA, y cada una de estas partes contiene los siguientes elementos:

- 1 único *convertidor de direcciones* o **ADRRESS CONVERTER**
- tantas *listas invertidas* (índices) como descriptores y superdescriptores se hayan definido en el archivo considerado.
- la *Tabla de gestión de almacenamiento* o **SMT** (Storage Management Table).

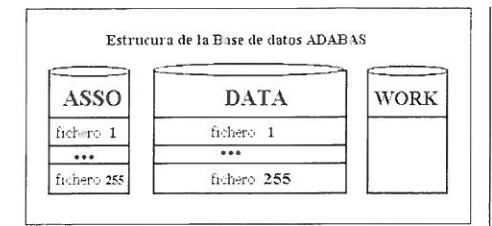


Figura 1.

Los siguientes apartados amplían estos conceptos que se acaban de enunciar.

ISN (INTERNAL SEQUENCE NUMBER)

Este es el elemento diferenciador de este modelo de SGDB. Es un **número secuencial** que el gestor asigna a cada uno de los registros en el momento de su alta en la base, y que le sirve para poder identificar al registro de datos.

Para poder asignar este número, el gestor consulta la tabla que contiene todos los ISNs asignados, el *Address Converter* o Convertidor de direcciones. Esta tabla, como ya se ha dicho, contiene en todos aquellos huecos correspondientes a ISNs utilizados, el *RABN* (Relative Address Bloc Number) del *DATA* en el que se encuentra ubicado el registro. Por tanto, solo tienen que buscar cual es el ultimo cuyo contenido sea distinto de cero, para saber cual es el ISN que debe asignar al nuevo registro. El bloque en el que ubicara el registro le asigna después de examinar la *SMT* (Storage Management Table - Tabla para la gestión del Almacenamiento) que le indican el espacio libre en cada bloque.

LISTAS INVERTIDAS:

Como ya se ha dicho al hablar del DATA, este gestor permite definir tantos índices como se consideren necesarios para poder recuperar la información almacenada en la Base de Datos. Cada uno de estos índices se denomina *DESCRIPTOR* si es campo elemental, o *SUPERDESCRIPTOR* si esta formado por la unión de varios campos.

Pues bien, una *lista invertida* no es mas que la forma que tienen ADABAS de implementar un índice definido sobre el fichero físico. Evidentemente, en base a lo anterior, la lista debe recoger todos los valores que estén contenidos en el campo correspondiente del registro del DATA, y, tiene asociada a cada una de las entradas, la lista de ISNs de los registros que contienen dicho valor. Además dispone de un contador que indica el número de ISNs asociados a dicho valor. Hay que tener en cuenta que los descriptores no tienen por que ser valores únicos; tal es el caso del campo descriptor PROVINCIA, de la figura 2.

Por tanto, se puede considerar que cada **DESCRIPTOR O SUPERDESCRIPTOR** definido genera una *LISTA INVERTIDA* con la siguiente estructura:

- Valor clave
- Número de ISNs que contienen el valor especificado,
- Lista de ISNs que tienen ese valor.

Ejemplo:

Con el fin de facilitar la comprensión de esta organización, se plantea el siguiente ejemplo:

Se desea realizar dos altas en el fichero de clientes sabiendo que cuando se definió físicamente, se estableció que Provincia fuera Descriptor, y que Teléfono también lo fuera.

Nombre	: Fulano	Mengano
Provincia	: Álava	Vizcaya
Teléfono	: 123.45.67	987.65.43

Pues bien, en el momento de dar de alta en la base a estos clientes, se asigna:

- al conjunto "fulano/ÁLAVA/1234567" el *ISN 28*, y se ubica el conjunto **"28/fulano/ÁLAVA/1234567"** en el RABN **12**.
- al conjunto "Mengano/VIZCAYA/9876543" el *ISN 29*, y se ubica el conjunto **"29/Mengano/28033/9876543"** en el RABN **12**.

Es el propio gestor quien, en el momento de almacenar el registro en la BD, no solo asigna el ISN, sino que además actualiza todas las listas invertidas del fichero, por lo que si no existe, en la lista invertida de Teléfono el valor contenido en el registro a almacenar, cosa normal pues en principio cada cliente tendrá su teléfono, genera una entrada en dicha lista invertida con los datos correspondientes, es decir

1234567	1	28
9876543	1	29

En cambio, a la hora de actualizar la lista invertida de Provincia, se encuentra con que ya existen clientes de Álava, por lo que se limita a incrementar el contador de ISNs de clientes de ÁLAVA, y a incluir el ISN 28 en la lista asociada a esta entrada; y hace lo mismo con la entrada correspondiente a VIZCAYA, con lo que resulta.

Álava	4	1,5,7,28
Vizcaya	3	2,6,29

Por su parte, en el *Address Converter*, el SGDB ha incluido en la casilla **28** el valor **12**, para indicar que el registro de datos se encuentra en el bloque 12, y en la casilla **29** también ha incluido el valor **12**, pues el segundo

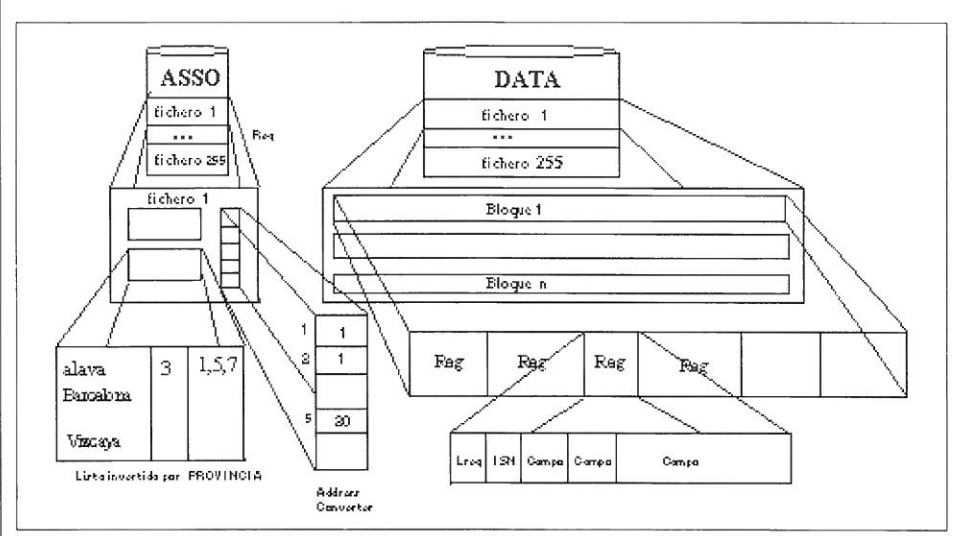


Figura 2.

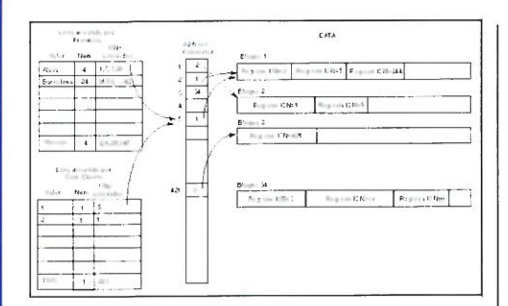


Figura 3.

registro también se encuentra en dicho bloque físico.

1	2
2	3
...	
28	12
29	12
...	

La figura 3 muestra esquemáticamente toda esta explicación.

Métodos de recuperación de información:

ADABAS permite recuperar la información mediante los siguientes métodos:

READ PHYSICAL de un fichero, en este caso ADABAS realiza la lectura secuencial del fichero de DATA, por lo cual lee el primer bloque, y devuelve el primer registro lógico del bloque físico, luego el segundo, y así hasta terminar el bloque, momento en el que lee el siguiente bloque, etc. ..., por lo que la información devuelta no sigue ningún orden.

READ by ISN: En este caso, ADABAS lee secuencialmente el convertidor de direcciones, es decir los ISNs del fichero, recuperando cada vez el bloque correspondiente a un ISN. Una vez cargado en memoria dicho bloque, localiza dentro del mismo el registro que tenga por ISN el ISN buscado. Para poder realizar esto, en el momento de darle de alta, coloca al principio del registro la **longitud** del mismo, de modo que si al examinar el ISN del primer registro del bloque, resulta que no es el buscado, salta los bytes correspondientes a dicho registro, y examina el ISN del segundo registro, y así sucesivamente hasta localizarlo, cosa que sin lugar a dudas consigue pues se encuentra en dicho bloque.

GET ISN: en este caso solo se pretende recuperar la información de un único registro del que se conoce su ISN. Para lo cual lo que hace el gestor es localizar mediante el Address Converter, el bloque en el que se encuentra el registro, leer dicho bloque y devolver al programa llamante los datos que ha solicitado de dicho registro.

READ by DESCRIPTOR provincia: En este caso, lo que hace ADABAS es recuperar la información de forma secuencial en base a la lista invertida que se especifique como descriptora para la recuperación. Es decir, va leyendo la lista invertida de PROVINCIA desde el principio o desde el valor que se desee. Una vez posicionado en la lista, va recuperando los registros asociados a cada valor de la lista en base a los ISNs que la acompaña, Así, para ÁLAVA, recupera primero el ISN 1, para lo cual se basa en el convertidor de direcciones que le indicara el bloque en el que se encuentra el registro asociado a dicho ISN. La siguiente vez que sea invocado ADABAS, devolverá el ISN 5, luego el 7, el 28, luego pasara a devolver los asociados a Barcelona, etc, hasta llagar al fin de la lista.

Los siguientes métodos no siguen una recuperación secuencial, sino que permiten extraer información con solo examinar las listas invertidas, buscando valores concretos:

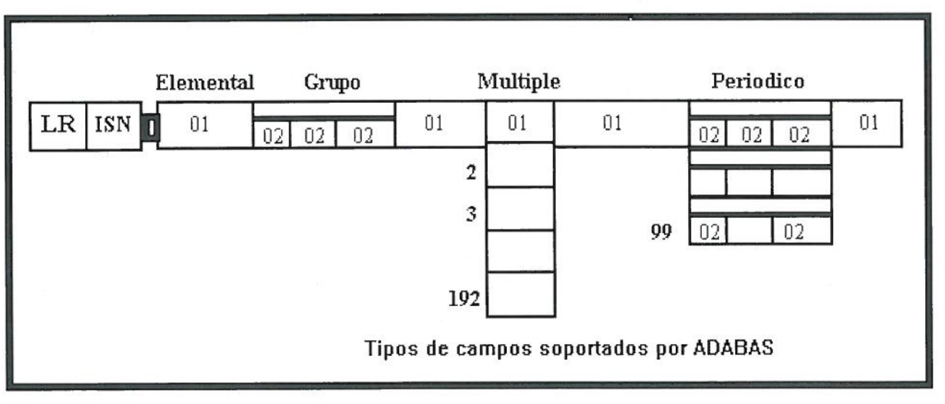
FIND with PROVINCIA=valor : En este caso, ADABAS examina y extrae de la lista invertida el conjunto de ISNs que cumplen la condición especificada. Y dicho conjunto (o **SET de ISNs**) le guarda en el WORK (el tercer fichero de la estructura que usa para su propia gestión). Una vez hecho esto, actúa normalmente, es decir, recupera el registro asociado al primer ISN del SET, Cuando se le pida el siguiente, ya no le buscara en las listas, sino que hará uso de este SET, por lo que devolverá el registro asociado al siguiente ISN, y así sucesivamente hasta que se acaben los ISNs del SET.

FIND with PROVINCIA=valor1 AND PROFESION=valor2: Este caso es parecido al anterior pues solo se ha añadido una condición a la sentencia de búsqueda. Con él se pretende mostrar la potencia del método, ya que permite determinar los registros que cumplen las dos condiciones, con solo examinar los ISNs asociados a los SETs de cada condición. Es decir, cuando ADABAS recibe esta petición, busca los ISNs que cumplen la primera condición, y les memoriza en su WORK, luego hace los mismo con la segunda condición, y por ultimo realiza la intersección entre ambos conjuntos, determinado los ISNs que cumplen las dos condiciones. Este es el set de ISNs buscados, y por lo tanto, una vez determinado solo tiene que ir recuperando los registros asociados a éstos para ir enviándoselos al programa llamante.

El número de condiciones que se pueden poner en una sentencia FIND es ilimitado, pero el programador debe ser consciente de la penalización que esto supone en el programa para la recuperación del primer registro.

FIND with PROVINCIA=valor1 AND PROFESION=valor2 sorted by NOMBRE:

Este es un caso mas que muestra la potencia del método. Una vez obtenido el set de ISNs que cumplen las dos condiciones, los registros deben ser devueltos en el orden que determine el descriptor usado como clasificador, en este caso **NOMBRE**. Esto es posible ya



Tipos de campos soportados por ADABAS

Figura 4.



que solo debe buscar los ISNs del SET resultante de las condiciones del FIND, entre los ISNs de la lista NOMBRE. Esta sentencia solo puede usarse con descriptores.

Además de estos métodos para la recuperación de los datos contenidos en los registros de la BD, ADABAS nos permite tratar las listas invertidas como si se trataran ficheros de datos mediante las sentencias siguientes:

HISTOGRAM provincia: En este caso, se lee la lista invertida de PROVINCIA recuperando la información en ella contenida. Es de resaltar que en este caso, solo se puede consultar los campos Valor-índice y *number que contiene el número de ISNs asociados a cada uno de los valores, es decir, ÁLAVA,3 BARCELONA,27 etc...

FIND NUMBER PROVINCIA=valor: Esta sentencia también nos recupera información de la lista invertida, devolviendo el valor del *number asociado al valor que se pasa como argumento. Así, si se hubiera dado ÁLAVA, devolvería 3.

Tipos de campos soportados por ADABAS:

Por ultimo es necesario describir los tipos de campos soportados por este gestor.

Campo ELEMENTAL: como su nombre indica es la unidad mínima de información.

Campo tipo GRUPO: Aquel que se descompone en varios campos elementales. Ejemplo: El campo NOMBRE_COMPLETO, se compone de los campos elementales NOMBRE, APELLIDO_1, APELLIDO_2.

Campo MULTIPLE: es un campo elemental que puede tener varias ocurrencias: Ejemplo: Supuesto que en el registro de empleados se desee mantener la información de los proyectos en los que ha participado cada empleado, este se definiría como múltiple, pudiendo tener hasta **192** ocurrencias cada campo múltiple.

Campo PERIODICO: Es parecido al múltiple, con la salvedad de que en lugar de ser un campo elemental, cada ocurrencia es un campo tipo grupo. Ejemplo, si queremos conocer en el caso anterior, no solo los proyectos en los que ha participado, sino las fechas de intervención, en lugar de definir un campo múltiple se definiría un campo PERIODICO, donde cada ocurrencia tendría como campos elementales NOMBRE_PROYECTO, FECHA_INICIO, FECHA_FIN.

LA INTEGRIDAD REFERENCIAL

SOFWARE A.G., la creadora de ADABAS, garantiza la integridad referencial de ADABAS mediante el uso de dos sentencias básicas para la realización del checkpoint: END TRANSACTION y BACKOUT TRANSACTION. La primera pasa a definitivas todas las modificaciones, altas y bajas realizadas por el usuario desde el ultimo checkpoint *sobre todos los ficheros actualizados*, y, mediante la segunda anula todas las modificaciones realizadas también sobre todos los ficheros desde el ultimo checkpoint realizado. Por tanto, debe ser el propio programador quien, mediante el correcto uso de estas instrucciones, pase a definitivas las actualizaciones de la B.D. generadas en la transacción.

la información contenida en la base de datos se almacena en los ficheros "DATA" y "ASSO"

DDM: MÓDULO DE DEFINICIÓN DE DATOS

Otra de las cuestiones planteadas en el pasado artículo al hablar de las ventajas de un sistema gestor, era el hecho de que al interponerse éste entre el programador y los datos, podía restringir el acceso a los mismos. Uno de los métodos usados para conseguir este objetivo es limitar la visión que el programador tenga del registro del fichero, y en consecuencia de los datos contenidos en el mismo. Esto se consigue definiendo varias "*vistas*" de un mismo registro. Así, por ejemplo, un vista del registro del fichero EMPLEADO contendría los datos básicos del mismo, tales como nombre, dirección, teléfono. Otra vista, también asociada al mismo fichero, contendría los datos económicos, tales como nombre, sueldo, primas, etc,

La DDM es precisamente la "vista" o módulo que usara el programador para acceder a los datos de un fichero. En el se encuentran sólo las definiciones de todos los campos accesibles del mismo junto con los atributos de tipo, nombre, formato y longitud, así como una indicación de los descriptores y superdescriptores.

ADABAS Y LAS FORMAS NORMALES:

Con sólo la enumeración de los tipos de campos soportados por ADABAS, en concreto por los campos múltiples y periódicos, puede parecer que esta base no se adapta a la *primera forma normal*. Pero en este punto es necesario resaltar que no es la Base de datos quien sigue las formas normales, sino el MODELO DE DATOS diseñado para soportar los datos de una aplicación concreta quien debe normalizarse..

La misma observación cabe hacer sobre las *2ª y 3ª forma normales*, pues como se ha descrito, los elementos no clave dependen no de una única clave, sino de todos los campos definidos como descriptores y superdescriptores, pues utilizando cualquiera de ellos se recupera la totalidad del registro (a través del ISN de la lista invertida).

Por último, cabe resaltar el hecho de que ADABAS permite implementar cualquier modelo de datos diseñado para bases de datos jerárquicas, en red, o relaciónales. Esto sólo es posible hacerlo con ADABAS por su arquitectura y por los múltiples métodos de recuperación de la información de que dispone.

CONCLUSIÓN:

El empuje protagonizado por ADABAS en las Grandes Instalaciones de España, avalan el posible exceso de entusiasmo mostrado por el autor de este artículo, pero es que combinando este gestor con *NATURAL*, lenguaje creado también por *SOFTWARE AG* para sacar el máximo partido a su BD, resulta una herramienta potente, sencilla de manejar, y capaz de permitir desarrollos entre 4 y 5 veces mas rápidos que con cualquier otro lenguaje.



PROGRAMACIÓN DE SONIDO

David Aparicio

Todos sabemos que gráficos y sonido son los aspectos más vistosos de un ordenador. Hace algún tiempo, vimos lo que se podía obtener en gráficos, con la librería para SuperVga en Linux. En este artículo, exploraremos las posibilidades del sonido en nuestro PC. Aunque disponer de una tarjeta de sonido es bastante corriente, los lectores que no dispongan de ella podrán todavía probar la programación del DSP con el altavoz interno, gracias al paquete de emulación que entregamos en el número anterior de la revista.

Los detalles del hardware, programación DMA, y otras oscuras maniobras a las que debemos estar acostumbrados desde DOS, desaparecen gracias al driver genérico VoxWare, que nos permite manejar de forma genérica nuestra tarjeta.Recordemos del número anterior que, en Unix, disponemos de dispositivos que nos permiten comunicarnos con el hardware, y que en el caso del sonido se agrupaban en /dev/mixer, /dev/audio, /dev/dsp y /dev/sequencer. El control sobre cada dispositivo se realiza con las primitivas read, write e ioctl. No seguiremos haciendo hincapié sobre este tema, sino que pasaremos a ver todo esto de forma práctica. Como /dev/audio y /dev/dsp se diferencian sólo en la forma logarítmica o lineal de captar el sonido, tomaremos la programación del segundo como un método común para ambos. Por tanto, en el resto del artículo analizaremos el mezclador, el DSP y el secuenciador FM/MIDI.

En DOS, cada programador debe tener su propia librería de sonido para sacar partido a su máquina. En Linux, todos los aspectos de bajo nivel están resueltos desde el mismo sistema operativo.

EL MEZCLADOR

El dispositivo /dev/mixer se encarga de captar todas las entradas y salidas de sonido que admite la tarjeta y controlar el volumen de cada una, así como si están presentes o no en el sonido final. En fin, se comporta como la mesa de mezclas de un estudio de grabación.

El primer problema con el que se encuentra un programa que dialoga con el manejador genérico es averiguar el número de líneas disponibles sin conocer la tarjeta de sonido que se encuentra instalada. Sería un error que el manejador nos permitiese conocer dicho tipo de tarjeta, porque ello nos podría tentar a "personalizar" el programa que construyamos según cada dispositivo. En su lugar, tenemos suficientes mecanismos para independizar dicho programa.

En el mezclador, todas las interacciones se realizan con la llamada ioctl, cuyo formato pasamos a describir:

ioctl(descriptor, función, &parámetros)

- El descriptor es el número que nos devuelve la llamada open, y se usa en cualquier llamada de C que se relacione con ficheros, asociando así el comando a invocar con el dispositivo deseado (en este caso, el mezclador).
- A continuación, un número de función, descrito en /usr/include/linux/soundcard.h, comunicará al manejador la operación que deseamos realizar.
- El último campo se rellena con un puntero al parámetro que se desea leer o escribir para realizar cada función.

Todas las constantes, macros y declaraciones necesarias para programar todos los dispositivos de sonido se obtienen incorporando la siguiente línea a nuestro programa:

#include <linux/soundcard.h>

Debemos asegurarnos que los dispositivos de sonido del directorio /dev existen y se encuentran con los permisos adecuados. Se puede automatizar este proceso con los comandos:

cd /dev
MAKEDEV audio
cat sndstat

El segundo comando sólo será necesario ejecutarlo una vez, probablemente durante la instalación de Linux. Por otro lado, debemos comprobar que el núcleo de Linux soporta sonido. Si el último comando no produce información sobre su tarjeta, deberá recompilar el núcleo, tal y como explicamos en un artículo anterior.

Finalmente, llegado este punto podemos empezar a programar el mezclador. Hemos incluido un programa de ejemplo, mixer.c, en el que nos basaremos a lo largo de este apartado.

Observando la figura 1, en primer lugar observamos la declaración de nombre_disp, con los nombres de los dispositivos que soporta la versión de VoxWare que se tenga en el momento de compilar el programa. Este artículo se ha basado en la 2.90r2, disponible en el núcleo 1.2.10 de Linux.
En el array se encuentran los dispositivos que puede controlar el manejador. Observe su utilización de nuevo en la figura 1:

- SOUND_MIXER_NRDEVICES: Es una constante que indica el número de dispositivos que puede controlar VoxWare en la versión que se esté utilizando. Sin embargo, dependiendo de la tarjeta que esté instalada, tendremos accesibles un subconjunto más o menos amplio de éstos. Las funciones con las que averiguamos el hardware disponible se describen a continuación.
- SOUND_MIXER_READ_DEVMASK: Función que devuelve en un entero un campo de bits con 1 en los dispositivos que se encuentran disponibles. El bit más a la derecha es el primer elemento del array. En el ejemplo, la variable mascara_disp guarda este resultado, y la macro ESTA_INSTALADO permite analizarla para saber la presencia de un dispositivo concreto.
- SOUND_MIXER_READ_RECMASK: En nuestro ejemplo, en mascara_rec almacenamos otro campo de bits con los dispositivos que admiten entrada, es decir, de los que se puede adquirir sonido. La macro que examinará esta variable es ES_GRABABLE.
- SOUND_MIXER_READ_RECSRC: Esta función interroga sobre el dispositivo de entrada que se encuentra activo, es decir, del que se leería actualmente para efectuar una grabación. Para nuestro ejemplo, mascara_rsrc es la variable que contiene esta información, y ES_ENTRADA la macro asociada.
- SOUND_MIXER_READ_STEREODEVS: Con esta llamada obtenemos un campo de bits que nos indica si cada dispositivo disponible es estéreo o no. Guardamos su valor en mascara_est, y lo examinamos con ES_ESTEREO.

Con dichas llamadas analizamos el número y tipo de líneas disponibles en nuestro mezclador. Finalmente, podemos actuar sobre el volumen asociado a cada dispositivo con las funciones:

- MIXER_READ(i): Permite obtener el volumen de la línea que corresponde al dispositivo "i". Su valor se encuentra en una escala de 0 a 100, independientemente de los valores reales que admita el hardware.
- MIXER_WRITE(i): Permite fijar el volumen a un valor concreto, también entre 0 y 100.
- SOUND_MIXER_WRITE_RECSRC: Permite alterar la línea de entrada de la que se adquirirán datos cuando se grabe.

De nuevo puede observar un detalle del uso en la figura 1. El programa de ejemplo se compila, como de costumbre, con:

cc -o mixer mixer.c

Al ejecutarlo sin parámetros, nos muestra el estado de todas las líneas. Aunque en otra tarjeta pueden aparecer más mo menos líneas, para una SoundBlaster Pro el resultado es similar a:

```
Dispositivos instalados:
    vol 90:90 estereo
  synth 75:75 estereo
    pcm 75:75 estereo
   line 75:75 estereo (rec off)
    mic   00 mono    (rec on )
     cd 75:75 estereo (rec off)
```

```
/* Declara los nombres de dispositivos disponibles
*/

char *nombre_disp[SOUND_MIXER_NRDEVICES]
= SOUND_DEVICE_NAMES;

...

/* Apertura del mezclador para consultar y modifi-
car */

  mixfp = open("/dev/mixer",O_RDWR);

/* Lectura de los dispositivos instalados */

  if(ioctl(mixfp,SOUND_MIXER_READ_DEVMASK
,&mascara_disp)==-1) {
    fprintf(stderr,"VoxWare no instalado o incompa-
tible\n");
    return 1;
    }

...

/* Selecciona el dispositivo activo para adquisicion
*/

  if(!ES_GRABABLE(i))
    fprintf(stderr,"*** El dispositivo no es de
entrada\n");
  mascara_rsrc |= (1 << i);
  ioctl(mixfp,SOUND_MIXER_WRITE_RECSRC,&
mascara_rsrc);

...

/* Rutina general de impresion del periferico "i" */

  ioctl(mixfp,MIXER_READ(i),&vol);
  printf("  %8s", nombre_disp[i]);
  if(ES_ESTEREO(i))
    printf(" %02i:%02i",vol%256,vol/256);
  else
    printf("    %02i",vol/256);
  printf(" %s%s%s\n",
        ES_ESTEREO(i)?"estereo ":"mono    ",
        ES_GRABABLE(i)?"(rec":"     ",
        ES_ENTRADA(i)?" on
)":ES_GRABABLE(i)?" off)":"    ");
```

El dispositivo "vol" equivale al volumen general de salida de la tarjeta. "synth" corresponde al volumen del dispositivo /dev/sequencer, y "pcm" al de /dev/dsp. Observamos también que los tres siguientes dispositivos son de adquisición, y corresponden a la línea de un amplificador externo, al micrófono o al CD-ROM. Todos los dispositivos son estéreo, excepto el micrófono, que es además la entrada activa. Para alterar el volumen del CD, por ejemplo, se utilizaría:

mixer cd 80:80

Dicho comando invocaría la función:

ioctl(mixfd,MIXER_WRITE(id("cd")),&val)

Por otro lado, para tomar el amplificador como señal activa de grabación, escribiríamos:

mixer +line

Cuya ejecución ,finalmente, produciría la llamada:

ioctl(mixfd,SOUND_MIXER_WRITE_REC SRC,&line)

Con este programa habremos explotado las posibilidades básicas del mezclador. En el código del programa podrá encontrar toda la información necesaria para fijar estos conceptos.

EL DSP

Con este dispositivo podremos muestrear y reproducir sonido digital, tal y como hace, por ejemplo, Windows con sus ficheros WAV. Su uso es extremadamente sencillo, y se limita a programar algunos parámetros antes de pasar a leer o escribir el propio sonido. Los parámetros se controlan mediante llamadas ioctl, disponiendo de las siguientes operaciones:

- SNDCTL_DSP_GETFMTS: Función que permite averiguar, mediante un mapa de bits, los tipos de muestras que admite nuestra tarjeta. Observe su uso en la figura 2.
- SNDCTL_DSP_GETBLKSIZE: Nos devuelve el tamaño del buffer (en bytes) que emplea el hardware para intercambiar datos con el sistema. Lo utilizaremos para reservar un bloque de memoria en nuestro programa, y en intercambiar datos entre el DSP y el disco. VoxWare se preocupa de tomar y liberar buffers internos adicionales para adquirir/reproducir las siguientes muestras mientras nosotros leemos o escribimos otras en el dispositivo, de forma que realiza la función de "doble buffer" y la propia gestión DMA de forma transparente para nosotros. De nuevo podemos observar su uso en la figura 2.
- SOUND_PCM_READ_BITS: Esta llamada nos indica el tipo de muestras que se utilizará en este momento para muestrear o reproducir. Se establece con la llamada gemela SOUND_PCM_WRITE_BITS.
- SOUND_PCM_READ_CHANNELS: Esto indica el número de canales, (1 para muestreo mono, y 2 para estéreo) que emplearemos para muestrear. Se puede variar con SOUND_PCM_WRITE_CHANNELS.
- SOUND_PCM_READ_RATE: Esta función dice la velocidad de muestreo o reproducción, en Hz. Se altera con SOUND_PCM_WRITE_RATE.

Tras definir las características con las que se adquiere/reproduce, podemos pasar a realizar en sí esta operación, con llamadas read ó write sobre el dispositivo /dev/dsp.

En el programa de ejemplo, dsp.c, se proporciona información sobre el tipo de onda activa, y los disponibles según nuestro hardware. Tras ello se informa de que se van a adquirir 5 segundos de audio. En este momento, podemos aprovechar el programa mixer, del apartado anterior, para seleccionar la entrada activa de la que grabaremos. Pulsando una tecla, procedemos a dicha adquisición. De nuevo se nos informa de que se va a reproducir el resultado. Empleando mixer de nuevo, podremos elegir el volumen de salida, y, pulsando otra tecla, escucharemos el resultado de la prueba.

Con este sencillo ejemplo, hemos mostrado los rudimentos de las operaciones con sonido digital. También adjuntamos el programa str, que permite reproducir ficheros con formato "mod" siguiendo el mismo principio. Se adjunta para poder estudiar su código fuente y terminar de comprender lo explicado aquí.

EL SECUENCIADOR

Por último, vamos a explorar la programación de un dispositivo MIDI. El /dev/sequencer comprende tanto este aspecto como el generador FM que incorporan prácticamente todas las tarjetas de sonido, y su manejo es idéntico. Aunque los ficheros de sonido relacionados con este modo son los que ocupan menor espacio en disco, son también los más complicados de obtener, puesto que se basa en una programación previa del sonido de cada instrumento (librerías de sonidos, o patches), para pasar luego a describir secuencias de notas y tiempos, interpretando efectivamente una partitura más o menos compleja.

Tarjetas de sonido con "síntesis por tabla de ondas" pueden ofrecer mayor riqueza de sonido que las que disponen de "sintentizador FM", notándose esta diferencia al cargar la definición de cada instrumento. En el primer caso, se pasa al manejador una serie de parámetros que definen ataque, caída, etc.

```
/* Deteccion de los modos de onda disponibles */
/* Dependiente de VoxWare 2.90r2 */
#define NMODOS 10

  char *nm_modos[] = {
    "Detectar",
    "Ley-mu",
    "Ley-A",
    "ADPCM",
    "8 bits sin signo",
    "16 bits (Intel) con signo",
    "16 bits (Motorola) con signo",
    "8 bits con signo",
    "16 bits (Intel) sin signo",
    "16 bits (Motorola) sin signo",
    "" };

  if(ioctl(dspfd,SNDCTL_DSP_GETFMTS,&
mascara_fmts)==-1) {
    fprintf(stderr,"VoxWare no instalado o
incompatible\n");
    return 1;
    }
  printf("Formatos soportados por DSP:\n");
  for(i=0; i< NMODOS; i++)
    if(mascara_fmts&(1 << i))
      printf("    %s\n",nm_modos[i+1]);
  printf("\n");

...

/* Reserva memoria y reproduce (sin cambiar
caracteristicas) */
/* Por defecto, VoxWare se programa para 8KHz
8 bits mono */
  if(ioctl(dspfd,SNDCTL_DSP_GETBLKSIZE,&
pcm_size)==-1) {
    fprintf(stderr,"VoxWare no instalado o
incompatible\n");
    return 1;
    }
  if((abuf = (void *)malloc(pcm_size)) == NULL) {
    fprintf(stderr,"No puedo reservar buffer de
sonido\n");
    return 2;
    }
  printf("Pulsa una tecla para reproducir...\n");
  getchar();
  if((fd = open(TMPNAME,O_RDWR))<0) {
    fprintf(stderr,"No puedo leer fichero de
sonido\n");
    return 2;
    }
  for(i=0; i<40000/pcm_size; i++) /* Calculo de 5
segundos a 8 KHz */
    write(dspfd,abuf,read(fd,abuf,pcm_size));
  close(fd);
```

En la síntesis, lo que define a cada instrumento es una muestra PCM del mismo.

En el programa de ejemplo, sequencer.c, se comienza por averiguar el número y tipo de dispositivos que están disponibles. Después, se carga un parche ejemplo, con la síntesis FM de un piano, como se puede observar en la figura 3. Se emplea la llamada ioctl sobre la función SNDCTL_FM_LOAD_INSTR.

Mediante otras llamadas más complejas, podríamos cargar patches para síntesis "wave", como en el caso de la GUS. Nótese que aquí interrogaríamos de forma más directa sobre el hardware instalado, para saber las capacidades que tenemos al alcance de nuestro programa, pero sin llegar a averiguar el tipo concreto de tarjeta.

```
/* Variables y estructuras utilizadas por las
MACROS MIDI */
/* La declaracion de este codigo es OBLIGATO-
RIA para usarlas */
int seqfd;
SEQ_DEFINEBUF(2048);

void seqbuf_dump(void) {
  if(_seqbufptr) {
    if(write(seqfd, _seqbuf, _seqbufptr) == -1)
{ fprintf(stderr,"Error de acceso a /dev/sequen-
cer\n");
      exit(1);
      }
    _seqbufptr = 0;
    }
  }

...

/* Informacion sobre la onda del instrumento */
const char Piano[16] =
 { 0x11,0x01,0x8A,0x40,0xF1,0xF1,0x11,0xB3,
   0x00,0x00,0x06,0x00,0x00,0x00,0x00,0x00 };

  /* Cargar PATCH de instrumento "Piano" */
  instr.key = FM_PATCH;
  instr.device = INSTR;
  instr.channel = CANAL;
  memcpy(instr.operators,Piano,16);
  ioctl(seqfd, SNDCTL_FM_LOAD_INSTR, &instr);

...

  /* Detalle de un comando MIDI de 4 bytes */
  /* Asociar #CANAL con instrumento #INSTR */
  /* se observa el uso de las variables de */
  /*  las macros VoxWare                   */
  _seqbuf[_seqbufptr++] = SEQ_PGMCHANGE;
  _seqbuf[_seqbufptr++] = CANAL;
  _seqbuf[_seqbufptr++] = INSTR;
  _seqbuf[_seqbufptr++] = 0;


  /* Esto podría haberse encapsulado en una
macro */
  /* inventada por nosotros, asi:          */
  SEQ_PGM_CHANGE(CANAL,INSTR);

...

const char nota[] = { 0,2,4,5,7,9,11,12 };

  /* Detalle de uso de macros para interpretar
notas */
  /* Ejecutar notas */
  for(i=0; i<8; i++) {
    SEQ_START_NOTE(INSTR,CANAL,nota[i]+
12*OCTAVA,VOL);
    SEQ_WAIT_TIME((i+1)*TEMPO);
    SEQ_STOP_NOTE(INSTR,CANAL,nota[i]+
12*OCTAVA,VOL);
    }
  /* Flush de la cola de notas sobre el dispositivo
*/
  SEQ_DUMPBUF();
```

La carga y gestión de los instrumentos que se emplearán se realiza en un módulo separado de VoxWare, denominado "patch manager", y que se encuentra todavía en desarrollo.

A partir de ahora, las operaciones sobre el secuenciador se limitan a la escritura de datos, con la llamada de librería write. Los datos que se envían tienen que ver con el estándar MIDI, y se basan en secuencias de 4 u 8 bytes, donde el primero representa un comando y el resto proporciona parámetros del mismo. El programa simplemente va completando un buffer con los comandos que desea que MIDI interprete, y lo vuelca en el dispositivo secuenciador cuando está lleno.

Para facilitar el manejo de comandos en nuestros programas, el fichero "soundcard.h" ofrece una serie de macros al efecto. Dada su variedad y especialización, su explicación está fuera de este artículo, y sólo mencionaremos las más básicas, reproduciendo una escala musical en nuestro ejemplo. El primer comando empleado es el SEQ_PGMCHANGE, que asocia el instrumento que hemos cargado con un canal de salida MIDI. Observamos en la figura 3 que estamos escribiendo sobre un buffer con un índice que se ha declarado internamente, en la macro SEQ_DEFINEBUF. Todas las macros que se describen se basan en que unos nombres de variables muy concretos, y para aprovechar dichas facilidades debemos restringirnos a dichos nombres.

Una vez realizado esto, las macros SEQ_START_NOTE y SEQ_STOP_NOTE indican el pulsado y soltado de qué teclas deseamos realizar para producir una cierta melodía. Disponemos de varios canales para producir sonidos simultáneos o polifónicos (como pulsar varias teclas a la vez en un piano). Por último, la macro SEQ_WAIT_TIME es imprescindible para contar el tiempo que debe durar cada nota a la hora de interpretar las partituras. De nuevo, consulte la figura 3 para ver un extracto de su uso en el programa.

Con este breve bosquejo, podemos ver la forma básica en la que se obtiene sonido sintetizado. El lector imaginará, como ya se ha comentado, que los ficheros de sonido resultantes son cortos, puesto que se pueden describir varios segundos de música con unos pocos bytes. Sin embargo, la dificultad de generar este tipo de sonido se basa en la "magia negra" que se debe emplear para averiguar los parámetros FM que corresponden a cada instrumento. En la síntesis por tabla, dicha información es mas sencilla de lograr, puesto que se basa en muestras PCM obtenidas de un sonido real.

Para una revisión completa de comandos, el lector interesado deberá consultar el fichero "soundcard.h", en el directorio /usr/include/linux.

CONCLUSIONES

Con esta sencillez podemos empezar a explorar las posibilidades de obtener sonido en Linux. Los programadores acostumbrados a manejar tarjetas de sonido en DOS pueden notar la patente diferencia entre ambos entornos. VoxWare todavía está lejos de estar completado, pero ofrece una estandarización que permite concentrarse en la aplicación, y no en el hardware.

Sin embargo, si usted tiene conocimientos profundos sobre la programación de una tarjeta concreta, su ayuda puede ser importante para contribuir al desarrollo del driver correspondiente, o para completar uno ya existente. Se soportan de forma nativa la SoundBlaster (Pro), SoundBlaster 16, Gravis UltraSound y ProAudio Spectrum, mientras que el resto de tarjetas deben ofrecer compatibilidad con alguna de las anteriores. Los fuentes del núcleo que contienen esta información se encuentran en el directorio /usr/src/linux/drivers/sound, y se estructuran, en base a lo que hemos repasado aquí, de la siguiente forma:

- mezclador: ics2101.c (gus), pas2_mixer.c y sb_mixer.c
- dsp: sb_dsp.c, sb16_dsp.c, pas2_pcm.c, ad1848.c (gus max) y gus_wave.c
- secuenciador: opl3.c, sequencer.c, sound_timer.c y sys_timer.c
- patch manager y midi: patmgr.c, mpu401.c, uart6850.c y ficheros *_midi.c

El resto de ficheros tratan aspectos genéricos del sonido, como la programación DMA (dmabuf.c), detección de hardware (*_card.c), y otros. Los puntos de entrada genéricos están en soundcard.c y sound_switch.c

Para cualquier consulta, el lector puede contactar con el autor por carta a la redacción, o bien por Internet, a la dirección "daparic@colibri.tid.es".

Dado que la robustez y flexibilidad del entorno Linux benefician a los programadores, esperamos que este artículo haya despertado su interés, y que contribuyan a la aparición de vistosas aplicaciones de sonido en nuestro sistema operativo preferido. O quien sabe, quizá una demo. Suerte.

PROGRAMACIÓN EN RED

Juan Manuel y Luis Martín

Posiblemente, uno de los campos de la microinformática que ha adquirido una mayor importancia y ha experimentado una mayor evolución son las comunicaciones e interconexión de ordenadores.

La consecuencia de esta gran evolución de las comunicaciones es el gran auge de las denominadas Redes de Área Local, consistentes en un conjunto de PCís interconectados entre sí, de forma que los usuarios puedan compartir tanto la información como los periféricos. Este tipo de redes ha permitido abaratar costes a las empresas y conseguir una conjunción en el funcionamiento de todos sus equipos.

Las Redes de Área Local son realmente beneficiosas para los usuarios finales, pero también conllevan un aumento de la complejidad en el desarrollo de aplicaciones. La principal preocupación que le surge al programador es que, en un entorno de red, sus programas no se están ejecutando sólos, sino que al mismo tiempo pueden estar ejecutándose otras aplicaciones. Y eso no es lo peor, sino que además, las otras aplicaciones pueden también acceder a sus ficheros de datos y alterar el contenido de los mismos.

Esta nueva situación ha llevado a los programadores a diseñar una serie de estrategias y herramientas encaminadas a ordenar los accesos a las bases de datos para que, de esta forma, no se produzcan situaciones erróneas en relación a los datos almacenados en dichas bases de datos.

ACCESOS CONCURRENTES

Cuando se trabaja en entornos multiusuario, todas aquellas operaciones que sucedan de forma simultánea se denominan concurrentes. De esta forma, cuando dos programas se están ejecutando en el mismo momento, se dice que se están ejecutando concurrentemente. Este hecho en sí no es el que más puede preocupar al programador cuando desarrolla una aplicación destinada a ejecutarse en un entorno de red. El aspecto más importante del desarrollo de aplicaciones para trabajar en entornos de red, por el peligro que conlleva para la integridad de los datos, es la compartición de bases de datos por varios programas que pueden ejecutarse de forma concurrente.

Esta situación puede dar lugar a que en un determinado momento, dos programas que se están ejecutando concurrentemente intenten acceder al mismo registro de una base de datos. Si dicho acceso concurrente tiene como fin la lectura del contenido de dicho registro por parte de los dos programas, no existe ningún problema, pero, ¿y si alguno de ellos (o ambos) intenta cambiar el contenido de dicho registro?. Si esta operación no se realiza de una forma controlada, es muy posible que se produzca una error, bien en las aplicaciones, bien en el contenido final del registro. Un ejemplo puede contribuir a aclarar el riesgo generado por esta situación.

Supóngase que el ordenador central de un banco contiene una base de datos con las cuentas corrientes de sus clientes. Aunque dicha base de datos se encuentra en el ordenador central del banco, todas las sucursales poseen ordenadores con un programa que permite acceder y cambiar el contenido de dicha base de datos. Podría ocurrir que en determinado momento la sucursal S1 del banco se disponga a abonar un cheque extendido por el cliente X, y en ese mismo momento, el cliente X se

encuentra ingresando una cierta cantidad de dinero en la sucursal S2 del banco. El programa de la sucursal S1 leerá el saldo de la cuenta, restará la cantidad indicada en el cheque y, finalmente, almacenará en el campo del saldo la nueva cantidad resultante. Por su parte, el programa de la sucursal S2 leerá la cantidad almacenada en el saldo, le sumara la cantidad a ingresar y, por último, almacenará la cantidad resultante de la operación.

Si ambas operaciones se realizan de forma secuencial no existe ningún problema. Pero, dado que el acceso se realiza de forma concurrente, es muy posible que esto no ocurra, por lo que el resultado final es impredecible. Un caso puede ser el que ambas sucursales lean la cantidad al mismo tiempo, y tras ello, cada una por su lado realice la operación indicada. A la hora de almacenar, ambas realizarán la operación de grabar el registro con cantidades diferentes, por lo que la cantidad final almacenada será errónea.

EL SISTEMA DE BLOQUEOS

Para poder evitar estos problemas, tanto Clipper, como la mayoría de aplicaciones orientadas al manejo de bases de datos, incorporan una serie de herramientas cuyo fin es el de impedir que varios usuarios puedan acceder a un mismo archivo o registro cuando alguno de ellos tiene intención de realizar alguna modificación en el mismo. Esta herramienta se denomina Sistema de Bloqueos o simplemente Bloqueos.

La misión de estos bloqueos es muy simple. Consiste básicamente en impedir (bloquear) el acceso a un archivo o registro determinado cuando se va a realizar una operación para la manipulación de los datos del mismo. De esta forma se evita que varios usuarios manipulen al mismo tiempo la información de una base de datos sin ningún control.

Por lo tanto, para actualizar o manipular la información almacenada en una base de datos o en un registro de la misma, lo primero que debe hacerse es bloquear, bien la base de datos completa o bien el registro concreto, según la prioridad de la operación a realizar. Cuando se haya conseguido el bloqueo podrán realizarse las operaciones que sean necesarias que, una vez finalizadas, permitirán desbloquearlo.

El sistema de bloqueos proporcionado por Clipper contiene varios niveles de protección que el programador debe elegir en cada momento, en función del tipo de operación a realizar. Estos niveles van desde el bloqueo completo de la base de datos durante toda la sesión de trabajo hasta el bloqueo de un registro determinado durante un cierto instante de tiempo. Por tanto, el programador debe manejar dos parámetros para la selección del tipo de bloqueo. Por un lado debe seleccionarse el nivel de bloqueo que garantice el correcto funcionamiento del sistema. Sin embargo, también debe tenerse en cuenta que en un entorno de red no se está trabajando sólo, por lo que debe procurarse mantener desbloqueado todo aquello que no sea necesario, con el fin de que pueda ser utilizado por el resto de los usuarios de la red.

APERTURA EXCLUSIVA O COMPARTIDA

El nivel de máxima protección contra posibles accesos concurrentes lo proporciona el proceso de apertura de una base de datos. En el momento de la apertura de una base de datos es posible especificar si ésta va a ser exclusivamente utilizada por nuestra aplicación, o si por el contrario se va a permitir que otras aplicaciones puedan también acceder a la base de datos.

USE xcBaseDatos ... EXCLUSIVE | SHARED
DBUSEAREA([lÁrea], [cControlador], cBaseDatos, [xcAlias], [lCompartido], [lSóloLectura])

Si la operación a realizar con la base de datos requiere un uso exclusivo de la misma (por ejemplo, cuando se desea hacer una operación de cambio de masivo en la información de un campo o para actualizar el contenido de la base de datos con información contenida en otros ficheros) debe indicarse la cláusula EXCLUSIVE en el mandato USE de apertura (o un valor .F. en el parámetro lCompartido de la función *DBUSEAREA()*). De este modo, la base de datos actualmente abierta solamente podrá ser utilizada por nuestra aplicación, y el

TABLA 1

Operación	Causa
USE ... SHARED	La base de datos está abierta en modo exclusivo por otro usuario
USE ... EXCLUSIVE	La base de datos está abierta en cualquier modo por otro usuario
APPEND BLANK	Otro usuario tiene el fichero bloqueado o ha ejecutado APPEND BLANK simultáneamente

Causas de errores de apertura de bases de datos.

resto de ususarios tendrá negado el acceso a la misma. Dicha base de datos ni si quiera podrá ser abierta por otra aplicación. Evidentemente, esta es la forma de trabajo más segura, pero al mismo tiempo la menos práctica de cara al trabajo en red. Para poder abrir la base de datos en modo exclusivo, ésta no puede encontrarse abierta previamente por ningún otro usuario en cualquiera de los modos.

Si por el contrario, se desea abrir la base de datos de forma compartida, es decir, permitiendo a los demás usuarios realizar operaciones con la misma, debe incluirse la cláusula SHARED (o un valor .T. en el parámetro lCompartido). Los bloqueos en una base de datos compartida se harán por otros medios y serán, por tanto, menos estrictos que el trabajo en modo exclusivo.

Por defecto, las bases de datos serán abiertas en modo compartido, ya que salvo raras excepciones, la mayoría de las operaciones no requieren un nivel de bloqueo tan alto.

Otra forma de selección del modo de apertura es mediante el mandato SET EXCLUSIVE. Mediante este mandato es posible indicar el modo de apertura de las bases de datos por defecto, es decir, cuando se omiten las cláusulas anteriormente mencionadas.

SET EXCLUSIVE ON | OFF | xlActivar

EL RESULTADO DE LA OPERACIÓN

Cuando se intenta abrir una base de datos en un entorno de red y ésta ya se encuentra abierta en modo exclusivo por otro usuario, o bien cuando se intenta abrir la base de datos en modo exclusivo y ésta ya se encuentra abier-

ta por otro usuario en cualquier modo, se produce un error de ejecución.

Estos errores, a diferencia de todos los demás, no producen la salida automática de la aplicación. Además, Clipper proporciona una función que permite verificar si el intento de apertura ha tenido éxito o no. Se trata de la función *NETERR()*, que devuelve un valor lógico indicando si se ha producido o no algún tipo de error durante la apertura.

Siempre que se intente abrir una base de datos en un entorno de red debe consultarse el valor devuelto por dicha función ya que, como se explicó anteriormente, los errores de bloqueo no producen la salida del programa, pudiendo dar lugar a errores más graves en el proceso de la información.

```
USE clientes EXCLUSIVE
IF !NETERR()
  ? "Ok!"
ELSE
  ? "Error de apertura"
ENDIF
```

La tabla 1 muestra las posibles causas que pueden provocar un error durante el proceso de apertura de una base de datos.

La única posibilidad poder desbloquear una base de datos abierta en modo exclusivo es cerrarla. Cuando se produce la operación de cierre de una base de datos, esta pierde los bloqueos indicados por el usuario que la había abierto.

BLOQUEAR UN FICHERO COMPARTIDO

Los siguientes niveles de bloqueo para poder controlar los accesos concurrentes a una base de datos no son tan estrictos y seguros como el visto anteriormente, pero aportan una mayor libertad de utilización de la base de datos por el resto de los usuarios de la red. Para utilizar estos niveles es necesario abrir la base de datos en modo compartido.

Si la operación a realizar involucra a varios de registros de una base de datos (por ejemplo, aquellas operaciones que incluyan las cláusulas WHILE, FOR o ámbito), siempre será conveniente bloquear el fichero entero. Si por el contrario, la operación a realizar sólo afecta a un registro concreto de la base de datos, únicamente será necesario bloquear dicho registro, dejando libres los demás para que puedan ser utilizados por el resto de usuarios.

El bloqueo temporal de un fichero completo que haya sido abierto en modo compartido se realiza mediante la función FLOCK(). Esta función permite bloquear una base de datos, siempre que ésta no se encuentre ya bloqueada, ni ninguno de sus registros. Además, esta función devuelve un valor lógico que indica si la operación ha tenido éxito.

Este tipo de bloqueo, a pesar de afectar a todo el fichero, es mucho menos rígido que el bloqueo producido por la apertura exclusiva ya que, cuando se bloquea una base de datos mediante esta función, el resto de usuarios pueden acceder a la misma para realizar operaciones de lectura, aunque nunca de escritura.

Si, por el contrario, lo que se desea es bloquear un único registro, será necesario posicionarse sobre él y ejecutar la función *RLOCK()*. Esta función bloquea el registro actual y devuelve un valor lógico indicando si la operación ha tenido éxito.

Cuando se bloquea un registro mediante el uso de la función *RLOCK()*, el resto de usuarios puede realizar cualquier tipo de operación con los demás registros de la base de datos. Además, el resto de usuarios también podrá realizar operaciones de lectura sobre el registro bloqueado.

En el caso de que un usuario tenga bloqueado el fichero mediante la función *FLOCK()* y él mismo ejecutase la función *RLOCK()*, el bloqueo sobre el fichero desaparece, pasando a estar bloqueado únicamente el registro actual.

Una vez realizadas las operaciones oportunas sobre el registro o fichero actualmente bloqueado, debe procederse al desbloqueo de los mismos, con el fin de permitir que otros usuarios puedan escribir nueva información sobre ellos. Esta operación se realiza mediante la función *DBUNLOCK()*. La ejecución de dicha función eliminará todos los bloqueos existentes sobre la base de datos del área de trabajo actual.

TABLA 2

Mandato	Bloqueo Mínimo
@...GET	RLOCK()
APPEND FROM	FLOCK()
DBDELETE()	RLOCK()
DELETE (Varios regs.)	FLOCK()
PACK	USE...EXCLUSIVE
DBRECALL()	RLOCK()
RECALL (Varios regs.)	FLOCK()
REINDEX	USE...EXLUSIVE
REPLACE (Un registro)	RLOCK()
REPLACE (Varios regs.)	FLOCK()
UPDATE ON	FLOCK()
ZAP	USE...EXCLUSIVE

Requisitos mínimos de bloqueo.

Si lo que se desea es eliminar los bloqueos de todas las bases de datos abiertas debe utilizarse la función *DBUNLOCKALL()*, cuya misión es idéntica a la función anterior, salvo que su ejecución afecta a todas las áreas de trabajo, no sólo a la actual.

En la tabla 2 pueden consultarse los requisitos mínimos de bloqueo para las operaciones más comunes con una base de datos.

INTENTOS DE BLOQUEO

Son múltiples las ocasiones en las que un bloqueo no funciona al primer intento. A raíz de este problema el programador debe preguntarse el número de ocasiones o el tiempo que debe seguir insistiendo para conseguir dicho bloqueo. El propio Clipper proporciona una solución a dicha cuestión, mediante la incorporación de una serie de funciones no estándar del lenguaje.

Estas funciones tienen la particularidad de incorporar un argumento especial de tipo numérico que permitirá indicar al programador el número de segundos durante los que desea seguir insistiendo en la operación de bloqueo.

Estas funciones realizan múltiples intentos consecutivos para conseguir el bloqueo de forma automática durante el tiempo especificado. Si se consigue realizar el bloqueo, dichas funciones devolverán un valor .T. al programa. Si, por el contrario, la función no ha conseguido dicho bloqueo una vez transcurrido el intervalo de tiempo especificado, se devuelve un valor .F. al programa.

Las funciones en cuestión son las siguientes:

- *NETUSE(cBaseDatos, lModo, nSegs)*: Intenta abrir una base de datos en

CORREO LECTORES

Para cualquier duda referente a los temas/artículos tratados en la revista, escriba una carta a:

Sólo Programadores
Referencia: *Correo Lectores*
TOWER COMMUNICATIONS
C/ Marqués de Portugalete, 10 Bajo
Madrid 28027

CÓMO SUSCRIBIRSE A





































Suscríbase enviando este cupón por correo o fax (91) 320.60.72, o llamando al teléfono (91) 741.26.62 Horario 9 a 14 y 15:30 a 18:30 h.

Deseo suscribirme a la revista SÓLO PROGRAMADORES acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año (12 números) **por sólo** 11.950 ptas. (ahorro 20%). ☐ Estudiantes carreras técnicas: 8.950 ptas. (ahorro 40%)

*** ESTA OFERTA ANULA LAS ANTERIORES, DESCUENTOS NO ACUMULABLES.**

Nombre y apellidos...Domicilio...

Población...C.P ....Provincia.................. ......Telf..............................Profesión....................

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº ____ ____ ____ ____

Fecha de caducidad de la tarjeta................Nombre del titular, si es distinto...

☐ Domiciliación bancaria.

Señor Director del banco ....

Población ....

Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta de ahorro número..

ENTIDAD	OFICINA	DC	Nº CUENTA

el recibo que le será presentado por TOWER COMMUNICATIONS, S.R.L. como pago de mi suscripción a la revista SÓLO PROGRAMADORES.

Firma:

☐ Contra-reembolso del importe más gastos de envío.

☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.

☐ Giro Postal (adjunto fotocopia del resguardo).

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Marques de Portugalete 10, Bajo
28027 Madrid.

modo exclusivo durante el intervalo de tiempo especificado en el argumento nSegs.

- *FILLOCK(nSegs)*. Intenta bloquear un fichero durante el intervalo de tiempo especificado en el argumento nSegs.
- *REGLOCK(nSegs)*. Intenta bloquear un registro durante el intervalo de tiempo especificado en el argumento nSegs.

Estas tres funciones son paralelas a *USE ... EXCLUSIVE, FLOCK()* y *RLOCK()* respectivamente. Como se ha indicado anteriormente, no se trata de funciones estándar, y son aportadas por Clipper a modo de ejemplo. Su código fuente puede obtenerse del el fichero LOCKS.PRG, que se encuentra situado en el directorio \CLIPPER5\SOURCE\SAMPLE. Para poder ser utilizadas en cualquier aplicación, será necesario compilar dicho fichero. Además, el fichero objeto resultante deberá ser indicado en la cláusula FILE del enlazador, con el fin de que el código sea incluido junto con el programa que las utiliza.

ADICIÓN DE REGISTROS

Una de las operaciones cuyo modo de funcionamiento ha variado con respecto a las versiones anteriores a la 5.2 es la de adición de un registro nuevo. Hasta ahora, la adición de un nuevo registro a una base de datos se realizaba de idéntica forma cuando se trabajaba en red o no. Para ello, Clipper dispone de un mandato y una función que realizaban la misma operación (APPEND BLANK y DBAPPEND()).

Sin embargo, a partir de la versión 5.2, la función *DBAPPEND()* ha alterado su forma de funcionamiento con el fin de adaptarse a la forma de trabajo en red. Para ello, se ha dotado a la función de un argumento de tipo lógico:

```
DBAPPEND( [lLiberar] )
```

Mediante el argumento lLiberar, el programador puede indicar si desea eliminar el resto de bloqueos que tenía sobre los demás registros de la base de datos. Esta operación puede parecer algo insignificante, pero el programador debe notar que, en la mayoría de las ocasiones, la introducción de los datos de un nuevo registro se realiza de forma manual, es decir, tecleándolos. Esta operación suele consumir un periodo bastante grande de tiempo, sobre todo en bases de datos con una gran cantidad de campos por registro. De esta forma, es posible liberar los registros bloqueados durante dicho tiempo para que el resto de usuarios puedan utilizarlos.

Son múltiples las ocasiones en las que un bloqueo no funciona al primer intento

Además, Clipper incorpora una función paralela en el fichero LOCKS.PRG para poder realizar varios intentos de adición de un nuevo registro, al igual que sucedía con las funciones vistas anteriormente para realizar los bloqueos. El nombre de esta función es *ADDREC()* y, al igual que las otras funciones, incorpora un único argumento numérico para indicar el número de segundos durante los cuales se desea seguir intentando realizar la operación.

```
IF !ADDREC( 5 )
  ? "No se pudo a–adir el registro"
THEN
  ? "Registro en blanco a–adido y bloqueado"
ENDIF
```

OTRAS OPERACIONES EN RED

Existen otro tipo de operaciones que deben ser realizadas con cierta precaución cuando se está trabajando en un entorno de red.

Una de las operaciones importantes a la hora de realizar un programa destinado a ser utilizado en un entorno de red consiste en le comprobación de que el ordenador se encuentra efectivamente conectado a la red, ya que, en caso de estar desconectado, las operaciones que dicho programa realiza encaminadas al trabajo en red pueden dar lugar a errores. Para ello, Clipper incorpora una función que, además, permite conocer el nombre de identificación de la estación de trabajo en la que se está ejecutando el programa. Se trata de la función *NETNAME()*. Esta función devuelve una cadena de caracteres que contiene el nombre de la estación de trabajo. Si el ordenador no se encuentra conectado a la red, la función devolverá una cadena nula.

```
cNombre := NETNAME()
IF EMPTY( cNombre )
  ? "Ordenador no conectado a la red"
ELSE
  ? "El nombre de la estación de trabajo es ", cNombre
ENDIF
```

Otra de las operaciones que pueden conllevar riesgos cuando se está trabajando en un entorno de red es la impresión de documentos. Ello es debido a que, por regla general, el acceso a la impresora se realiza por medio de la red, y puede dar lugar a conflictos con el resto de usuarios.

Para evitar este tipo de conflictos y conseguir un mayor nivel de seguridad en el funcionamiento del programa, así como para mantener la integridad de los documentos que se desean imprimir, es recomendable realizar una copia de los mismos en soporte magnético. Para realizar esta operación basta con redirecionar la salida de la impresora a un fichero de texto. De esta forma, aunque surja algún problema con la red a la hora de imprimir, el documento está almacenado en un fichero de texto. Una vez concluida la operación de almacenamiento de los documentos en dicho fichero, puede procederse a la impresión del mismo.

Por último, a la hora de utilizar controladores RDD dentro de un programa destinado a trabajar en un entorno de red debe tenerse en cuenta que algunos de ellos pueden presentar problemas en función de los programas y utilidades que se encuentren accediendo de forma concurrente a las bases de datos (programas como dBASE, FoxPro, Acces, etc).

JUEGOS CONVERSACIONALES

Ignacio Cea

Una vez desarrollado el modelo de objetos inicial (entrega anterior) hemos de, primero pulirla, y segundo, empezar a distribuir sobre las clases que lo forman las responsabilidades que creamos convenientes para resolver nuestro problema, que recordamos, era construir un conjunto de clases comunes a cualquier juego conversacional.

Además, aprevecharemos esta ocasión para, por un lado, explicarle algunos conceptos nuevos como el de FrameWork o el de Idioma y, por otro, mostrarle de forma práctica como el empleo de técnicas orientadas a objetos en el desarrollo de sistemas facilita la inclusión de cambios y modificaciones.

LA EVOLUCIÓN DEL MODELO

El modelo de objetos que describimos en la anterior entrega puede serle muy útil a la hora de tener una visión global de las clases que intervienen dentro de cualquier juego conversacional y de la forma en la que aquellas se relacionan entre si; por tanto, es muy importante que no lo pierda de vista.

Es necesario recordarle, sin embargo, que ese modelo de objetos no es algo único e inmutable. A poco que usted haya pensado sobre el tema habrá deducido no sólo nuevas clases que puedan intervenir en él, sino quizás, en todo un nuevo modelo de objetos que también resuelva nuestro problema. No se alarme, es algo natural; de hecho, nosotros mismos se lo demostraremos evolucionando un poco nuestro modelo inicial de partida.

IDIOMAS Y FRAMEWORKS

Por FrameWork entendemos aquel conjunto de clases relacionadas entre sí que pueden ser empleadas por cualquier usuario para resolver un problema concreto de programación.

Por ejemplo, todos sabemos que cualquiera de los compiladores actuales llevan siempre incorporadas un conjunto de clases que permiten el desarrollo de aplicaciones bajo entornos gráficos como OS2 o Windows. Con sólo redefinir levemente algunas de esas clases (herencia) conseguimos un sistema que se adaptará íntegramente a la mayor parte de nuestras necesidades.

La forma en la que esas clases deben ser redefinidas o usadas, o incluso relacionadas con otras, es algo perfectamente establecido por el FrameWork (no se puede crear una ventana de usuario derivando, por ejemplo, de la clase Button). Esas pautas nos obligan a desarrollar los sistemas siguiendo una especie de reglas no escritas. Esas reglas son, precisamente, lo que dentro del mundillo del C++, se denominan Idiomas.

LAS CLASES ABSTRACTAS

Recuerde que una clase abstracta era aquella que tenía, al menos, un método virtual puro. Esto significa, básicamente, que para poder usar esa clase el usuario debe redefinir esos métodos en alguna otra hija de la anterior. Por tanto, podemos concluir que declarando un método virtual puro dentro de una clases estamos estableciendo, de forma implícita, una pauta para su uso concreto. Las clases abstractas van a ser, en definitiva, la base de nuestro FrameWork.

El resultado final de esta serie de artículos será algo que, por si sólo, no haga nada. Será el usuario final el que tenga que redefinir el FrameWork para adaptarlo a sus necesidades. No se preocupe también haremos un ejemplo de ello. Un FrameWork es eso, un marco de trabajo; un idioma.

En la anterior entrega nos ocupamos de describir, a muy grandes rasgos, cuales eran los principales requerimientos del problema que tratábamos de resolver. A partir de este momento pasaremos a describir con más detalle el conjunto de clases que participan dentro de un juego conversacional. No pierda detalle.

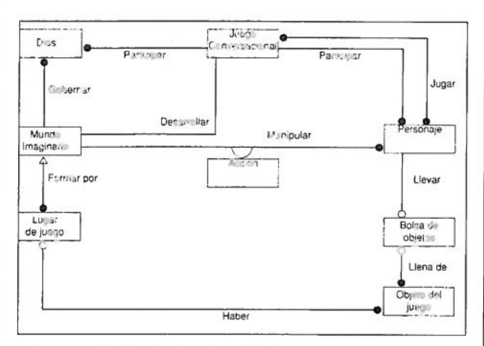


Figura 1. Modelo de objetos nuevo.

FIGURA 2

```
//==========================================================================//
// CLASE  :          JuegoConversacional                        //
// FECHA  :          12 de Octubre de 1995                              //
// DESCRIPCION:      Controla la ejecución de toda una partida       //
// AUTOR  :          Ignacio Cea Forniés                             //
//==========================================================================//

        #ifndef __JCONVERS_HH__
        #define __JCONVERS_HH__

        #include "Lista.hh"

        class Dios;
        class MundoImaginario;
        class Personaje;

        class AlmacenJuego;

class JuegoConversacional
{

public :

//——-*** Contructores/Destructores ***————————————————-//
JuegoConversacional (const String &);
virtual ~JuegoConversacional ();

//——-*** Métodos de acceso ***——————————————————-//
MundoImaginario *mundo (void);
const Lista<Dios *> &dioses (void) const;
const Lista<Personaje *> &jugadores (void) const;
Personaje *quien_esta_jugando (void);
Personaje *ganador (void);
AlmacenJuego *almacen (void);

//——-*** Manejo de la partida ***————————————————-//
void siguiente_jugador (void);
void anterior_jugador (void);
virtual Personaje *jugar (void);

protected :

MundoImaginario *ElMundo;                  // ¿Dónde se desarrolla el juego?
Lista<Dios *> LosDioses;                   // ¿Qué dioses gobiernan el mundo?
Lista<Personaje *> LosJugadores;           // ¿Quién juega?

AlmacenJuego *ElAlmacen;                   // ¿Dónde se almacena el juego?
Personaje *ElJugadorActivo;                // ¿A quién le toca?
Personaje *ElGanador;                              // ¿Quién ha ganado?

};

        #endif

// Fin del fichero
```

Clase Juego Conversacional.

LA GRAN PAUTA: NO CERRARSE

Si de verdad se pretende que el framework sea algo versátil no podemos cerrarnos en su diseño. No podemos, por tanto, hacer cosas como limitar el número de acciones que puede desarrollar un personaje o el tipo de catástrofes que pueden suceder sobre el mundo. Todo debe ser redefinible.

Lo mejor que podemos hacer es comenzar a expandir y completar en esa dirección nuestro modelo de objetos inicial.

DIOS SÍ JUEGA A LOS DADOS

No parece, al menos a primera vista, que podamos observar claramente dentro del modelo de objetos algo que pueda producir juegos conversacionales en los que la incertidumbre y aleatoriedad sean parte integrante del mismo. Ya que existe una clase llamada "Juego Conversacional" la cual parece hacer las veces de gestor integral del juego, esa tarea habría que asignársela a ella.

Las acciones a desarrollar como parte de esa aleatoriedad no están, sin embargo, definidas. Debe ser el usuario de la clase quien las defina creando una hija de ella. Podría decirse que a nuestro idioma le faltan algunas frases hechas. Esto debe mitigarse procurando un camino que guíe al usuario en la definición de la incertidumbre.

Para ello vamos a considerar un dios que gobierne el mundo y que, a su libre albedrío actúe sobre éste de la forma que estime más oportuna. La incertidumbre ya tiene brazo ejecutor.

DIOS A LO BESTIA

Para concretar la aleatoriedad basta, por tanto, con redefinir nuestro dios (un dios que provoque guerras, otro que resucite muertos, etc...). Sin embargo, sigue sin estar clara cual es la manera en la que debe implementarse ese incertidumbre. Es como si a nuestro dios le faltara un arma con la que trabajar. Pues bien, vamos a dársela en la forma de catástrofe natural.

Las catástrofes naturales tienen efectos sobre el mundo gobernado. Cada dios ha de elegir (aleatoriamente) que partes de ese mundo (Lugares de Juego) habrán de verse afectadas por la hecatombe que, a su vez, también habrán elegido al azar de entre todo un conjunto de ellas. Nuestro Idioma empieza a estar más completo.

OTRO PUNTO DE VISTA

Quizás se pregunte por qué considerar una clase como "Catástrofe Natural" cuando podríamos haberla sustituido por una serie de métodos como guerra, tormenta, diluvio o

FIGURA 3

```
//======================================================================//
// CLASE :          MundoImaginario                                    //
// FECHA :          12 de Octubre de 1995                              //
// DESCRIPCION      :          Lugar donde se desarrolla la acción del juego   //
// AUTOR :          Ignacio Cea Forniés                                //
//======================================================================//

          #ifndef __MUNIMAG_HH__
          #define __MUNIMAG_HH__

          #include "FechHora.hh"
          #include "RAccion.hh"

          class JuegoConversacional;
          class CatastrofeNatural;

class MundoImaginario
{

public :

//——-*** Constructores/Destructores ***——————————————————//
MundoImaginario (const String &,JuegoConversacional *);
virtual ~MundoImaginario ();

//——-*** Métodos de acceso ***——————————————————————-//
const String &nombre (void) const;
const FechaHora &fecha_y_hora (void) const;
JuegoConversacional *juego (void);

//——-*** Cambios continúos en el mundo ***———————————————-//
virtual ResultadoAccion girar (void) = 0;

//——-*** Catástrofes naturales ***————————————————————//
virtual ResultadoAccion catastrofe_natural (const CatastrofeNatural &) = 0;
virtual ResultadoAccion catastrofe_natural_sobre (const CatastrofeNatural &,
                    const Lista<String> &) = 0;

protected :

String ElNombre;
FechaHora LaFechayHora;

JuegoConversacional *ElJuego;

};

          #endif

// Fin del fichero
```

Clase Mundo Imaginario.

inundación dentro de la clase Mundo Imaginario que tuvieran el mismo efecto sobre él.

Es cierto que podría haberse hecho de esa manera pero, para crecer en el tipo de cosas que pueden afectar al mundo sería necesario tocar la clase "MundoImaginario" que va a ir definida dentro de una librería. Más adelante lo verá más claro. Observe los dos modelos para crecer.

SIN TRAUMAS

Seguro que ha caído ya en la cuenta de lo diferente que es tratar un problema mediante técnicas orientadas a objeto. No se ha hecho referencia ni una sola vez a la forma en la que el juego se va a desarrollar; algo que, por el contrario, hubiera sido primordial con otro tipo de enfoque. Sólo se han apuntado entidades y formas de relación entre ellas pero nada sobre lo que han de ejecutar ni en el orden el que han de hacerlo.

También hemos alterado profundamente el diagrama de objetos inicial de una forma más bien sencilla y sin provocar grandes quebraderos de cabeza. La realimentación es algo que en orientación a objetos se hace patente.

UN ALTO EN EL CAMINO

El conjunto de clases que intervienen dentro de nuestro FrameWork final es enorme (muchas más de las que se pueden observar dentro de este diagrama y que ya iremos tratando) y tratar de abordarlas de forma simultánea no puede ser más que el caos más absoluto.

Dentro de este artículo vamos a ocuparnos de ver las cuatro clases que parecen estar en la parte superior del conjunto; esto es: "Dios", "Juego Conversacional", "Mundo Imaginario" y "Catástrofe Natural".

Antes de proseguir nos gustaría que se percatara de un pequeño detalle. Los nombres de las clases dentro del modelo de objetos no pueden ser implementados tal cual dentro de C++ ya que, algunos de ellos, comprenden más de dos palabras. El modelo de objetos es algo que se utiliza para comprender el problema y no debe tratar de relacionarse, inicialmente, con el lenguaje en el que va a ser implementado. Por eso desde estas líneas aconsejamos emplear un lenguaje natural (acentos, preposiciones, etc...) dentro del modelo de objetos y no el típicamente informático (sin acentos ni preposiciones ni etc...). Esta, sin duda, es una opinión del autor del artículo y somos conscientes de que otros autores opinan en diferentes términos.

Por último, añadir, que la entidad dios va estar reflejada en nuestro FrameWork como la clase Dios y que el hecho de emplear aquí mayúsculas y no minúsculas no tiene la menor intención de ofensa sino el seguir la norma de nomenclatura de clases adoptada desde un principio. Disculpen si alguien se sintió ofendido.

JUEGO CONVERSACIONAL

La clase Juego Conversacional tiene atribuidas dos responsabilidades fundamentales: jugar una partida y

mantener un almacén (disco) de todos los elementos que participan en ella. De esta última nos ocuparemos en posteriores entregas.

El juego conversacional ha de tener conocimiento de, primero, cual es el mundo en el que tiene lugar la acción y segundo de quienes son tanto los jugadores que participan de forma activa en el juego como de los dioses que gobiernan dicho mundo.

Jugar una partida significa llevar el control sobre ella, permitiendo que, en unas ocasiones, los dioses actúen sobre el mundo mediante catástrofes naturales y que en el resto, sean los personajes quienes lo modifiquen a través de sus acciones. Además debe procurar que el mundo gire.

El método fundamental, por tanto, es "jugar". En la mayor parte de las ocasiones, las aplicaciones conversacionales tendrán un sólo objeto de este tipo que estará ejecutando de forma cerrada el método "jugar" hasta que la partida finalice. Sin embargo, esto es algo que debemos dejar a la elección del usuario de la clase, por tanto incluimos dentro de ella métodos que nos permiten conocer que jugador es el activo, quien ha ganado, cambiar al siguiente, al anterior, etc... Observe la figura alusiva esta clase.

FIGURA 4

```
//========================================================================//
// CLASE :          Dios                                                  //
// FECHA :          12 de Octubre de 1995                                 //
// DESCRIPCION      :       Personaje ocupado de gobernar el mundo y sus  //
//                          elementos naturales.                          //
// AUTOR :          Ignacio Cea Forniés                                   //
//========================================================================//

        #ifndef __DIOS_HH__
        #define __DIOS_HH__

        #include "RAccion.hh"

        class JuegoConversacional;
        class MundoImaginario;

class Dios
{

public :

//——-*** Contructores/Destructores ***——————————————//
Dios (const String &,JuegoConversacional *);
virtual ~Dios (void);

//——-*** Métodos de acceso ***——————————————//
const String &nombre (void) const;
JuegoConversacional *juego (void);
MundoImaginario *mundo (void)

//——-*** Actúa sobre el mundo ***——————————————//
virtual ResultadoAccion actuar (void) = 0;

protected :

String ElNombre;                    // ¿Cómo se llama nuestro Dios?

JuegoConversacional *ElJuego;   // ¿En qué juego estamos participando?
MundoImaginario *ElMundo;         // ¿Sobre quien gobiernan? (Derivada)

};

        #endif

// Fin del fichero
```

Clase Dios.

DIOS

"Dios" sólo se ocupa de gobernar. Cuando "Juego Conversacional" estima que es momento ordena a los dioses que actúen sobre el mundo y estos decidirán si, primero, es el momento o no oportuno para ello, segundo cual es la "Catástrofe Natural" que van a emplear y, por último, que remotos parajes del mundo gobernado se verán afectados por ella.

La acción peso se ha propagado dentro de un objeto a todos sus miembros

Por la definición intrínseca de dios, no parece muy íntegro que sea otra clase quien le diga cuando actuar. dios debía funcionar de forma paralela al resto del juego. Sin embargo, estamos pensando que el FrameWork se desarrolle dentro de una máquina monotarea en la que la concurrencia ha de ser simulada.

El método central de "Dios" es "actuar" y cada dios ha de redefinirlo para adaptarlo a sus propias exigencias (Marte, Júpiter, etc...).

El efecto de la catástrofe en si sobre el mundo es algo que depende de la catástrofe en si; es decir, si hay un terremoto las casas se caen y eso es algo que no depende del dios que origina la catástrofe. Tenga mucho cuidado al distribuir las responsabilidades pues pueden echar a pique un buen FrameWork.

MUNDO IMAGINARIO

Nuestro "Mundo Imaginario" se encarga, básicamente, de "girar". Al igual que la actuación de los dioses, esto también habría de ser una acción que se desarrollara paralelamente al resto del juego; sin embargo, los motivos para que sea dispara por "Juego Conversacional" vuelven a ser los mismos.

"Mundo imaginario" de trasladar a todos los lugares elegidos por los dio-

FIGURA 5

```
//======================================================================//
// CLASE :          CatástrofeNatural                                   //
// FECHA :          15 de Octubre de 1995                               //
// DESCRIPCION  :   Desgracia que afecta a los persanajes y objetos     //
//                  que participan en el juego.                         //
// AUTOR :          Ignacio Cea Forniés                                 //
//======================================================================//

        #ifndef _CNATURAL_HH_
        #define _CNATURAL_HH__

        #include "String.hh"
        #include "Lista.hh"

        class ObjetoJuego;
        class Personaje;

class CatastrofeNatural
{

public :

//——-*** Contructores/Destructores ***————————————————//
CatastrofeNatural (const String &);
~CatastrofeNatural (void);

//——-*** Métodos de acceso ***————————————————————————//
const String &nombre (void) const;

//——-*** Efectos sobre el juego ***———————————————————//
virtual ResultadoAccion efectos_sobre (const Lista<ObjetoJuego *> &) = 0;
virtual ResultadoAccion efectos_sobre (const Lista<Personaje *> &) = 0;

protected :

String ElNombre;

};

        #endif

// Fin del fichero
```

Clase Catastrofe Natural.

ses el efecto de la catástrofe señalada. Eso se realiza a través del método "catástrofe natural sobre".

El pasar referencias y no objetos en sí favorece enormemente al rendimiento del conjunto

CATÁSTROFE NATURAL

La catástrofe natural afecta a los lugares y personas de un paraje. Esto se realiza mediante el método "afectar".

PROPAGACIÓN Y CONTAINERS

Dentro de la orientación a objetos en general y el C++ en particular no son extraños los casos en los que una acción aplicada sobre un objeto concreto se propaga a todas y cada una de las partes que la forman.

Para saber, por ejemplo, el peso de la "Bolsa de Objetos" de un personaje cualquiera del juego he de invocar a su método "peso". Esa llamada se traducirá en una llamada igual a todos y cada uno de los objetos que están dentro de la bolsa. El resultado de cada una de esas invocaciones se irá sumando a un acumulador que, en definitiva, será el valor retornado.

Podríamos decir que la acción peso se ha propagado dentro de un objeto a todos sus miembros. Este concepto es algo que se repite mucho dentro de las aplicaciones orientadas a objetos y en nuestro ejemplo particular lo podemos encontrar, entre otros sitios con los métodos "girar" y "catástrofe natural sobre" de la clase "Mundo Imaginario".

Estas situaciones suelen darse abundantemente dentro de aquellas clases que contengan instancias de otras, las cuales reciben típicamente la denominación de Containers.

OJO CON LO QUE LEE

Algunas de las figuras de este artículo muestran los ficheros de definición de las clases en el tratadas (Mundo Imaginario, Dios, Juego Conversacional y Catástrofe Natural).

Dentro de ellas aparecen referencias a muchas otras clases de las que, hasta ahora, no se ha hecho ni la más mínima mención. Tal es el caso de "Lista", "String", "ResultadoAcción" o "AlmacenJuego". Son clases necesarias para la implementación final del FrameWork y su papel dentro de él será desvelado a lo largo de próximas entregas. De hecho ni siquiera se suministran como código.

Emplee dichas figuras como guía aclarativa de lo expuesto o como guía de referencia de lo que ha de venir. Sea un poco paciente.

REFERENCIAS

Observe, también, el casi abusivo uso que se hace de las referencias en el paso de parámetros a métodos. El pasar referencias y no objetos en sí favorece enormemente al rendimiento del conjunto y es, siempre, algo a tener muy en cuenta.

PRÓXIMA ENTREGA

En la próxima entrega iremos ahondando en los detalles de algunas otras clases. Vaya pensado, mientras tanto, en la forma en la que cree que se detallarán y en la manera en la que puede llegar a emplear las ya vistas.

importante tener en cuenta que el valor es alfanumérico, puesto que la ordenación será alfabética y no numérica, esto es: 1, 10, 2, 20... y no 1, 2, 10, 20. Se suele escoger un nombre para el grupo de temas y a continuación un número, es buena idea no poner números consecutivos, para en un futuro poder insertar temas. Para el grupo de temas sobre gráficos del ejemplo que acompaña a este artículo se ha establecido la siguiente secuencia: Gráficos:0010, Gráficos:0020, Gráficos:0030...

REGIONES SIN SCROLL

Las regiones sin scroll son las que no muestran la barra de desplazamiento en la ventana de la ayuda. Esta parte de la ventana no se desplaza cuando desplazamos el texto hacia arriba.

Únicamente puede haber una región sin scroll en cada tema, inmediatamente debajo de la barra de botones, si se intenta definir otra, el compilador HC31.EXE dará un error. Las regiones sin scroll ayudan a mantener el título del tema visible, y pueden incluirse gráficos e hipergráficos, a los que se podrá acceder aunque se desplace el texto de la ventana. Para definir una región sin scroll hay que activar Conservar con el siguiente, en Presentación, de la opción Párrafo del menú Formato, en MS-Word 6, como muestra la figura 2.

APARIENCIA DE LA VENTANA DE LA AYUDA

En la sección [WINDOWS] del archivo de proyecto se define los atributos de la ventana de la ayuda. Se pueden, también, definir ventanas secundarias. El nombre interno de la ventana principal de la ayuda es Main, la sintaxis es la siguiente:

Main = "Título", (x, y, ancho, alto), maximizar, (color de fondo), (color de región sin scroll)

Main puede sustituirse por el nombre de otra ventana secundaria que se desee crear, en el ejemplo que acompaña a este artículo se ha definido la ventana secundaria WIN2, que se utiliza para mostrar la explicación de que

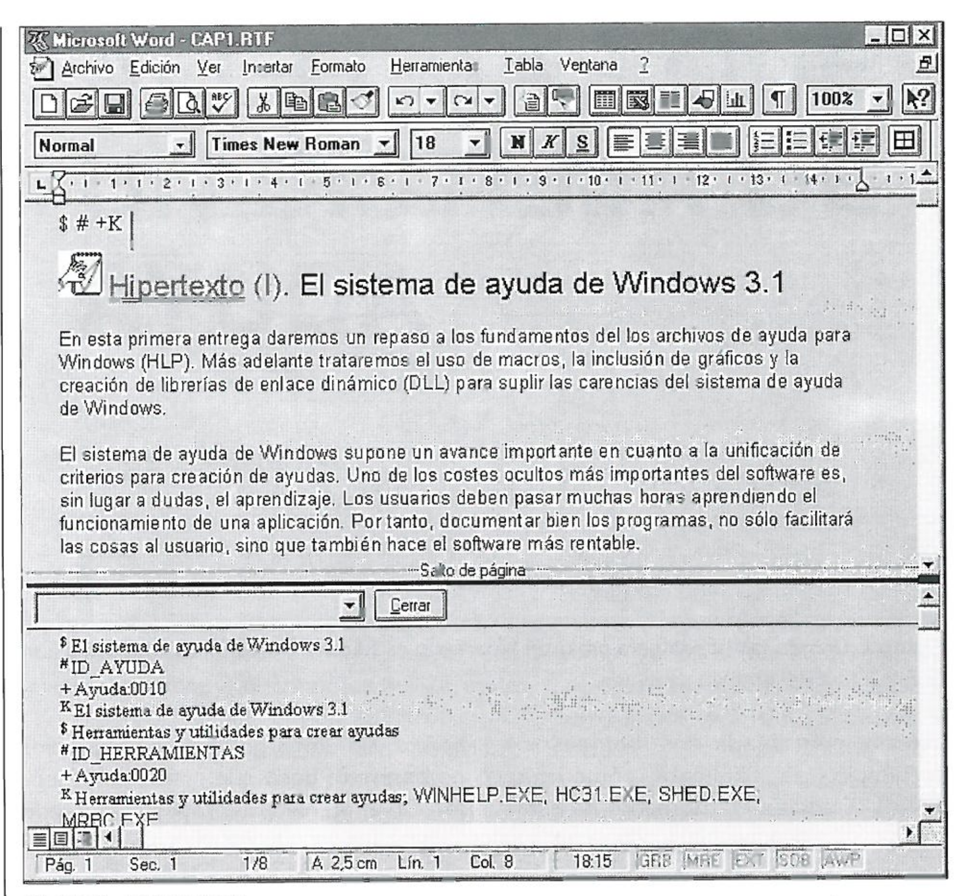
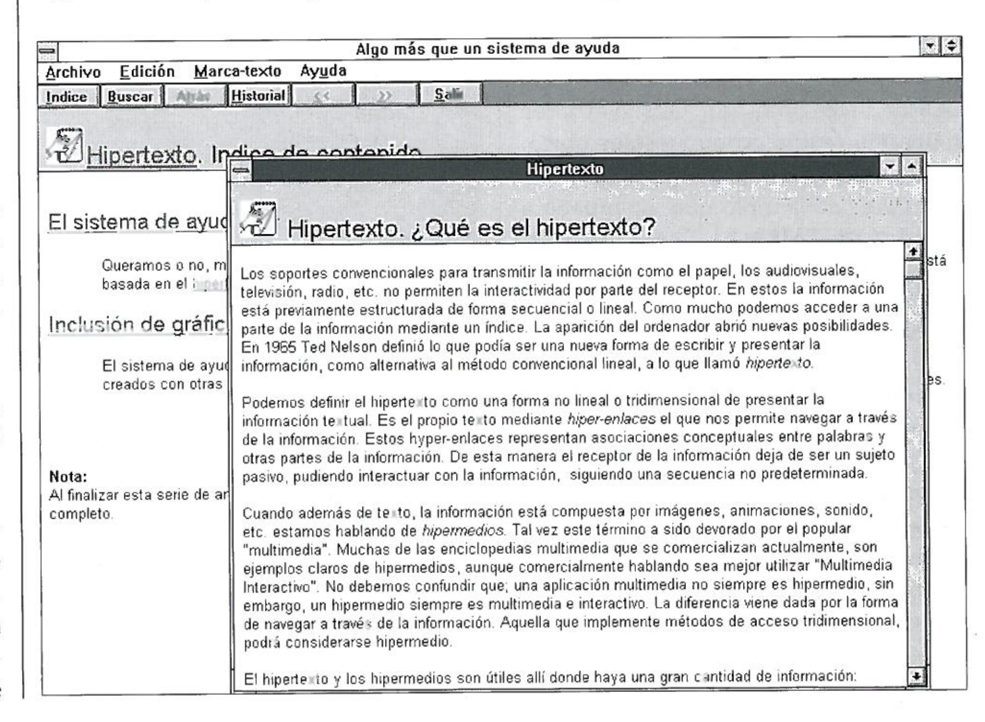


Figura 3. Archivo RTF.

es el hipertexto. Título es el título de la ventana, para el caso de Main, si ya se estableció en la sección [OPTIONS] mediante la opción TITLE, no será necesario ponerla, si se hace, prevalece sobre la definida en TITLE. Lo siguiente es la posición y tamaño de la ventana, en unidades de la ayuda de Windows, sea cual sea la resolución del monitor, el valor máximo para ancho y alto es 1023, si se pretende que la ventana ocupe la mitad de la pantalla el valor deberá ser 512. Los valores posibles para maximizar son 0



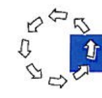


FIGURA 4

Menú	ID
Menú Archivo	MNU_FILE
Menú Edición	MNU_EDIT
Menú Marcar	MNU_BOOKMARK
Menú Ayuda	MNU_HELP

Identificadores de los menús de la Ayuda.

para ventana normal y 1 para presentar la ventana maximizada. Color de fondo es el color de la ventana de la ayuda en valores RGB, por ejemplo: (255, 255, 255) establece que el color de fondo será blanco y de igual modo para el color de la región sin scroll.

El usuario agradecerá que la opción buscar este activa, el procedimiento para hacerlo es sencillo

AÑADIR LA OPCIÓN BUSCAR

El símbolo de nota al pie (K) nos permite definir palabras clave que aparecerán en el cuadro Buscar de la ayuda. El procedimiento es simple, únicamente insertar al principio del tema la nota al pie (K) y escribir la lista de palabras clave del tema, separadas por un punto y coma. El botón de Búsqueda de la barra de botones de la ayuda se activa automáticamente. Como puede verse en el ejemplo que acompaña a este artículo.

MACROS PARA EL TRATAMIENTO DE MENÚS

El sistema de ayuda de Windows posee varias macros para el tratamiento de menús. A continuación se explican algunas:

InsertMenu("ID_MENU", "nombreMenú ", posición). Inserta un menú en la posición indicada en posición. ID_MENU es el identificador que se le dará al menú y nombreMenu es el nombre que tendrá en menú.

AppendItem("ID_MENU", "ID_OPCION", "nombreOpción", "macro"). Añade una opción al final del menú indicado en ID_MENU. ID_OPCION será el identificador de la opción de menú, que se podrá utilizar posteriormente con otras macros como DeleteItem. El nombre de la opción de menú se indica en nombreOpción, el símbolo (&) antes de una letra del nombre indica la tecla aceleradora. Macro es la macro que ejecutará la opción de menú. Los identificadores correspondientes a las opciones de menú por defecto de la ayuda de Windows, se muestran en la figura 4.

InsertItem("ID_MENU", "ID_OPCION", "nombreOpción", "macro", posición). Igual que AppendItem, pero permite indicar la posición del menú.
- DeleteItem("ID_OPCION"). Borra una opción de menú, ID_OPCIÓN es el identificador de la opción. No se pueden borrar las opciones de menú por defecto de la ayuda.

DisableItem("ID_OPCION"). Desactiva una opción de menú, ID_OPCIÓN es el identificador de la opción.

EnableItem ("ID_OPCION"). Activa una opción de menú, ID_OPCIÓN es el identificador de la opción.

CheckItem ("ID_OPCION"). Marca una opción de menú, ID_OPCIÓN es el identificador de la opción.

- UncheckItem ("ID_OPCION"). Quita la marca de una opción de menú, ID_OPCIÓN es el identificador de la opción.

EL EJEMPLO QUE ACOMPAÑA ESTE ARTICULO

Por falta de espacio en el disquete no se ha incluido el hipertexto completo. Se realizará uno que incluirá también posteriores entregas de esta serie de artículos, que podrá utilizar como manual de referencia del sistema de ayuda de Windows. Cuando finalice esta serie de artículos, permanezca atento a la siguiente edición de Sólo Programadores que incluya CD-ROM.

En el grupo de temas del ejemplo referente a la inclusión de gráficos, no se han implementado los hiper-enlaces ni las palabras clave para la opción buscar. Además no se han incluido hipergráficos, explicados en la entrega anterior. Sería un buen ejercicio terminar de implementar los hiper-enlaces y añadir hypergráficos al hipertexto. Una idea sería añadir una serie de hipergráficos en la región sin scroll para presentar una lista de temas relacionados o, desplazarse de un grupo a otro de temas.

Es muy importante que el lector de Sólo Programadores no se limite a leer mensualmente artículo a artículo esta serie, si no que trate de avanzar por su propia cuenta, utilizando lo aquí mostrado por el autor y lo que ya se ha entregado en anteriores disquetes de la revista.

AHORA ¡SAQUELE EL MAXIMO PARTIDO A SU MICROSOFT® WINDOWS 95!

Con las primeras aplicaciones de 32 bits para Windows 95: Microsoft® Office, Microsoft® Word, Excel, PowerPoint y Access.

Aproveche nuestras ofertas especiales de actualización para Usuarios Registrados Microsoft.

Pídalo ya en los Centros de Actualización Microsoft o en su distribuidor habitual.

Para más información llámenos al telf.: (91) 804 00 96





PROG.

EL DISEÑO DE UN LENGUAJE DE PROGRAMACIÓN

Daniel Navarro

En el anterior artículo de esta serie se vieron los conceptos más importantes relacionados con el mundo de los compiladores y los lenguajes de programación, para describir después de una forma global el funcionamiento de un compilador.

Cuando se va a construir un compilador para un lenguaje, lo primero siempre es aprender bien el lenguaje, o si es nuevo, concebirlo bien, con todos sus detalles. Se pretende en esta entrega abordar el diseño del lenguaje de esta serie: "Letra", así como, citar las principales características del entorno de programación. Se verán ejemplos que pongan de manifiesto las posibilidades del lenguaje que se va a implementar.

Los motivos por los que se plantea el diseño de un nuevo lenguaje son múltiples: crear un lenguaje específico para resolver un tipo concreto de problemas, crear un lenguaje para una nueva arquitectura, para soportar una nueva filosofía de programación (como orientada a objetos), para simplificar la entrada de datos a una aplicación (un lenguaje para definir los ficheros de configuración), para el manejo de cualquier tipo de maquinaria, o simplemente, por el placer de diseñar el mejor de todos los lenguajes creados hasta la fecha. "Letra" se ha creado con un fin educativo: enseñar a crear nuevos lenguajes y servir como base para ello.

La complejidad de un lenguaje viene determinada por el número de elementos que lo componen, y la flexibilidad de los mismos. El lenguaje que en esta serie se propone es muy sencillo, ya que su finalidad así lo requiere.

El diseño de un lenguaje es algo muy complejo; se debe tener en cuenta fundamentalmente dos factores: la finalidad del lenguaje (qué se va a programar con él) y quién lo va a utilizar.

Se ha tratado de crear un lenguaje con características comunes a los lenguajes de programación actualmente más utilizados, sin tender a ninguno en especial, manteniendo en todo momento la máxima simplicidad sin perder su funcionalidad. Prácticamente la totalidad de los lenguajes han sido diseñados en lengua inglesa, así que Letra será uno de los primeros diseñados en nuestra lengua, el castellano.

EL LENGUAJE LETRA

Letra es una simplificación de lenguajes de programación como Pascal, C o BASIC.

Las principales características de este lenguaje son:

-Se trata de un lenguaje de propósito general, ideado con un doble fin: servir como lenguaje para la iniciación a la programación y servir como base para la construcción de otros compiladores más complejos.

- Es un lenguaje con un solo tipo de variables, o datos como se denominarán a partir de ahora. Tan solo hay datos normales (variables sencillas) y tablas de datos (de una sola dimensión). Estos datos pueden ser utilizados indistintamente como de tipo carácter, por ejemplo para recoger el valor de una tecla, como de tipo número entero, para un contador, o de tipo lógico, expresiones boolean que sólo pueden tomar dos valores; cierto o falso.

- Todas los datos son de alcance global (como en la programación BASIC), es decir, todos los datos definidos en alguna parte del progra-

LISTADO 1

```
; Los comentarios se definen con un punto y coma

DATOS   ;Definición de los datos del programa

a
b=32
tabla1[4]
tabla2[4]=0:1:2:3

FIN

si a>0   ;Ejemplo de una sentencia tipo 'if'
  a=a-1
fin

si a>0 y b<0   ;Ejemplo tipo 'if .. else'
  a=b
sino
  b=a
fin

mientras b>0   ;Ejemplo tipo 'while .. do'
  b=b-2
fin

repetir   ;Ejemplo 'do .. while' o 'repeat .. until'
  a=a+2
hasta a>0

tabla1[a/2]=tabla2[b+a]   ;Ejemplo con tablas

hacer .pon_b_a_cero   ;Llamada a una función

volver   ;Fin del programa

.pon_b_a_cero   ;Ejemplo de una función

b=0

volver   ;Retorno de la función
```

Un programa en letra.

ma pueden ser usados en cualquier otra. Pero a diferencia de BASIC, todos los datos usados en el programa se deben declarar (como en Pascal o C).

- No se maneja ningún tipo de estructura de datos complicada, ni gestión de memoria dinámica, ni tablas multidimensionales, ni punteros, ni registros (o 'struct'), ni tipos enumerados, ni siquiera se utilizan símbolos separadores de sentencias (como el símbolo ';' de C).

- Los procedimientos (funciones o rutinas), son similares a como se programan en BASIC, pudiendo llamar a un procedimiento y volver de él, pero sin tener la posibilidad de pasarle parámetros locales (sólo se le puede pasar información en los datos globales).

- Las sentencias de control son las más típicas ('if', 'while ... do', 'do ... while' o 'repeat .. until', 'goto', 'gosub', 'return'), si bien en Letra, estas sentencias se construirán en castellano (ver en el cuadro 1 la sintaxis de las sentencias de control disponibles en Letra).

CUADRO 1

** SENTENCIA CONDICIONAL (SI ... FIN)*

Una sentencia condicional permite hacer una comprobación y según resulte, ejecutar o no una parte del programa. La comprobación se hace con una expresión lógica: si esta resulta cierta se ejecutará el código indicado. La sentencia se codifica de esta manera:

```
SI <Expresión lógica>
  ;Código para cuando <Expresión lógica> sea cierta
FIN
```

Opcionalmente se puede ampliar esta sentencia para ejecutar otro código cuando <Expresión lógica> sea falsa, quedando la sentencia así.

```
SI <Expresión lógica>
  ;Código para cuando <Expresión lógica> sea cierta
SINO
  ;Código para cuando <Expresión lógica> sea falsa
FIN
```

No es necesario poner <Expresión lógica> entre paréntesis como en otros lenguajes

Las zonas de código (código especificado para cuando la expresión lógica sea cierta o para cuando sea falsa) son porciones de programa que pueden contener cualquier número de sentencias, comentarios, zonas de datos, etiquetas, etc., siendo posible, por tanto, incluir aquí nuevas sentencias condicionales.

* SENTENCIA MIENTRAS (MIENTRAS...FIN).

Una sentencia mientras permite repetir iterativamente una parte del programa mientras una expresión .lógica se evalúe como cierta. Su construcción respeta el siguiente esquema:

```
MIENTRAS <Expresión lógica>
  ;Parte del programa que se repite
FIN
```

Si la expresión lógica es falsa, la parte de código contenida en la sentencia no se ejecutará. Asimismo, si el código contenido en la sentencia es incapaz de modificar el resultado de la expresión lógica el programa entrará en un bucle sin fin.

El código interior a la sentencia puede contener cualquier construcción del lenguaje, pudiendo por ello, anidar sentencias mientras.

* SENTENCIA REPETIR (REPETIR...HASTA).

Esta sentencia permite repetir iterativamente una parte del programa hasta que la expresión lógica se evalúe como cierta. Se construye respetando el siguiente esquema:

```
REPETIR
  ;parte del programa que se repite
HASTA <Expresión lógica>
```

Aunque la expresión lógica sea cierta, la parte de código contenida en la sentencia se ejecutará al menos una vez. Asimismo, si el código contenido en la sentencia es incapaz de modificar el resultado de la expresión lógica el programa entrará en un bucle sin fin.

El código interior a la sentencia puede contener cualquier construcción del lenguaje, pudiendo por ello, anidar sentencias repetir.

* SENTENCIA HACER / VOLVER.

La sentencia HACER transfiere el control del programa a la parte representada por la etiqueta especificada hasta la llegada de una sentencia VOLVER. La sintaxis de estas sentencias es como sigue:

```
HACER <nombre de etiqueta>
...
<nombre de etiqueta>
...
VOLVER
```

* SENTENCIA IR.

Se trata de un salto incondicional a la etiqueta indicada. La sintaxis de esta sentencia es la siguiente:

```
IR <nombre de etiqueta>
```

Sentencias de control.

UN PROGRAMA EN LETRA

En el listado 1 podemos observar un programa en Letra, que no realiza ninguna función en concreto, salvo servir de ejemplo. En él se puede observar la

forma que tiene un programa en este lenguaje. Otras características importantes del lenguaje que se deben resaltar son:

- No se distinguen en el lenguaje mayúsculas y minúsculas, ni para los identificadores de datos ni para las palabras reservadas ('CONTA' equivale a 'conta').

- Todos los datos, se denominarán (como en el resto de los lenguajes) por su identificador, que será un carácter alfabético seguido opcionalmente por una cadena de caracteres alfanuméricos (por ejemplo, podrá llamarse a los datos 'A', 'A23', 'a99lfa' o 'midato').

- Las tablas se definirán especificando entre corchetes, tras el identificador, el número de posiciones de estas y se utilizarán indicando la posición deseada (índice) en una expresión, entre corchetes y a continuación del identificador, (podremos definir una tabla de 20 datos como 'mitabla1[20]' y acceder a ella posteriormente con expresiones del tipo 'mitabla[i+1]'). Es posible inicializar una tabla en su declaración; para ello se deben indicar los valores separados por el símbolo ':'. Por ejemplo si se quisiera definir una tabla denominada 'palabra' de 5 elementos que coincidan con las letras (sus códigos ASCII se entiende) que componen la palabra 'Letra', se podría hacer de cualquiera de las siguientes formas:

palabra[5]='L':'e':'t':'r':'a'
palabra[5]="Letra"
palabra[5]=76:101:116:114:97

No es necesario indicar el número de elementos de la tabla explícitamente, ya que al especificar su contenido indirectamente se esta definiendo (en los tres ejemplos anteriores se podría haber omitido la constante 5).

- Un programa puede tener varias zonas en las que se definan datos (estas zonas comienzan con la palabra reservada 'DATOS', finalizando la definición de estos al llegar la palabra reservada 'FIN'); dichas zonas pueden ir en cualquier parte del programa, no siendo necesario declarar un dato antes de usarlo (aunque si es necesario declarar todos los datos que se usen).

- Es posible poner varias sentencias o declaraciones de datos en la misma línea, siempre que se separen por, al menos, un espacio en blanco.

- Los procedimientos o funciones no existen como tales, pero se puede hacer llamadas a un trozo de código poniendo al inicio de éste, una etiqueta (que es un identificador, similar a los identificadores de datos, pero que comienza con un punto) y la palabra reservada 'VOLVER' (similar a un 'return') al final del código.

Entonces cada vez que se quiera invocar ese código se puede hacer

CUADRO 2

OPERADORES GENÉRICOS

(	Abrir paréntesis, inicia subexpresión.
)	Cerrar paréntesis, finaliza subexpresión.

OPERADORES NUMÉRICOS

+	Signo más, operador aditivo, suma.
-	Signo menos, substracción, resta.
*	Operador multiplicativo.
/	División de enteros, cociente de división.
%	Módulo de la división o resto.

OPERADORES LÓGICOS

=	Operador de comparación, igual a.
<>	Distinto de.
>	Mayor que.
>=	Mayor o igual que.
<	Menor que.
<=	Menor o igual que.
Y	'AND', cierto si ambas expr. son ciertas.
O	'OR', cierto si alguna expresión es cierta.
NO	'NOT', cierto si la expresión es falsa.

En el cálculo de una expresión se realizan primero las operaciones con operadores de mayor precedencia, dentro de estos con una asociatividad de izquierda a derecha. La precedencia de operadores de mayor a menor es la siguiente:

( ) {los paréntesis tienen la mayor precedencia}
NO + - {signos más y menos, unarios}
* / %
+ - {operadores de suma y resta, binarios}
= <> > >= < <=
Y O {los operadores lógicos 'Y','O' tienen la menor precedencia}

Operadores.

CUADRO 3

EXPRESIÓN	TIPO	RESULTADO
3	numérica	3
1-999	numérica	-998
'A'	numérica	65
(conta+1)/(conta+1)	numérica	1
'A'+2	numérica	67
CIERTO	lógica	cierto
NO FALSO	lógica	cierto
3>2+1	lógica	falso
4=2+2 Y FALSO	lógica	falso
(a<=b*2) O (b+b=2*b)	lógica	cierto

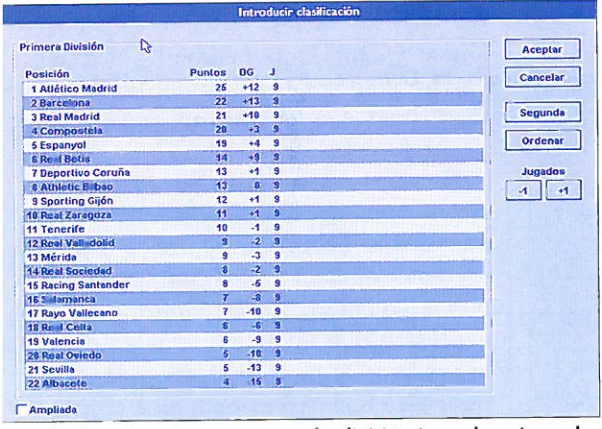
Expresiones.

SUPER QUINIELAS 3.0

TRES TEMPORADAS FABRICANDO MILLONARIOS

1993 el comienzo de Super Quinielas.

1994 segunda versión del programa.



Introducir clasificación

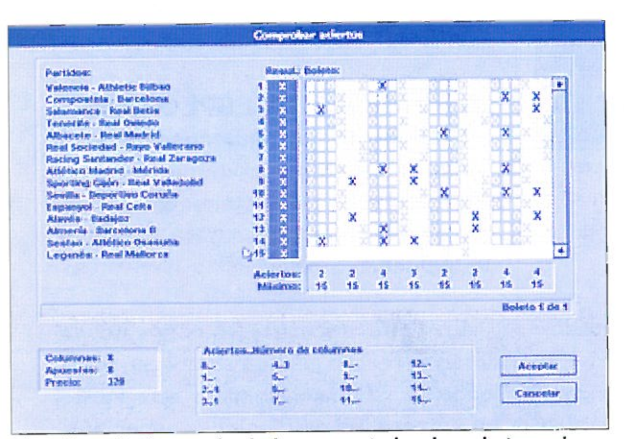
Primera División

Posición	Puntos	DG	J
1 Atlético Madrid	26	+12	9
2 Barcelona	22	+13	9
3 Real Madrid	21	+10	9
4 Compostela	20	+3	9
5 Espanyol	19	+4	9
6 Real Betis	14	+9	9
7 Deportivo Coruña	13	+1	9
8 Athletic Bilbao	13	0	9
9 Sporting Gijón	12	+1	9
10 Real Zaragoza	11	+1	9
11 Tenerife	10	-1	9
12 Real Valladolid	9	-2	9
13 Mérida	9	-3	9
14 Real Sociedad	8	-2	9
15 Racing Santander	8	-5	9
16 Salamanca	7	-8	9
17 Rayo Vallecano	7	-10	9
18 Real Celta	6	-6	9
19 Valencia	6	-9	9
20 Real Oviedo	5	-10	9
21 Sevilla	5	-13	9
22 Albacete	4	-16	9

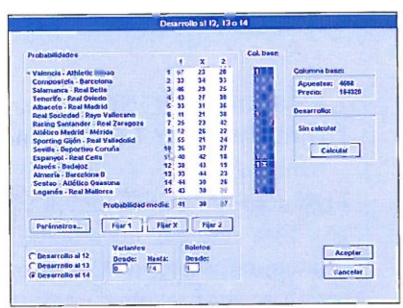
Aceptar · Cancelar · Segunda · Ordenar · Jugados -1 +1 · Ampliada

Siga la liga de primera y segunda división jornada a jornada, con las opciones de Super Quinielas.





Identifique fácilmente los boletos premiados de cada jornada.



Posibilidad de desarrollos al 12,13 ó14.

Base de datos de las últimas cinco ligas.

- *Tres sistemas de cálculo diferentes: histórico, últimos resultados, y por clasificación actual.*
- *Reducción al 12, 13 y 14.*
- *Base de datos de consulta de los últimos 5 años: primera y segunda división.*
- *Compruebe sus boletos premiados de la manera más rápida.*
- *Ideal para las peñas quinielísticas.*
- *Permite apostar mediante disquete.*
- *Válido para los próximos años / opción de nueva liga.*
- *Dos versiones por el precio de una: DOS y WINDOWS*

CON LA GARANTÍA DE DDM DIGIT AL DREAMS MULTIMEDIA

Solicita SUPER QUINIELAS 3.0 enviando este cupón o llamando al teléfono (91) 741.26.62 de 9 a 14 y de 15:30 a 18:30, o por Fax: (91) 320 60 72

Deseo que me envíen: ☐ **SUPER QUINIELAS VERSIÓN 3.0 por 2495 ptas. + 250 ptas. de gastos de envío.**

Nombre y apellidos.......................................Domicilio.......................................Población.......................

Provincia.......................................C.P ....F. de nacimiento.......................Profesión.......................................

FORMA DE PAGO:

Talón a ABETO EDITORIAL ☐ Contra-reembolso Teléfono .... Firma ,

☐ Giro Postal nº.......................................de fecha.......................

☐ Tarjeta de crédito VISA nº ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐

Fecha de caducidad de la tarjeta.......................................Nombre del titular, si es distinto.......................................

ABETO

Rellena este cupón y envíalo a: ABETO EDITORIAL C/ Marques de Portugalete 10, Bajo 28027 Madrid.

CUADRO 4

Borrar_pantalla()

Todos los programas inician su ejecución con la pantalla en modo texto 80x25 y borrada, pero se puede volver a borrarla llamando a esta función. Los programas borran automáticamente la pantalla al finalizar su ejecución.

Escribir("Cualquier texto ...")

Esta función recibe como parámetro un literal (un texto entre comillas) y lo saca por pantalla a partir de la posición del cursor actual.

Leer_tecla(<dato>)

Si se ha pulsado alguna tecla esta función deja en <dato> su valor ASCII, en caso contrario la función devuelve un 0 en <dato>. Si cuando se llama a la función hay varias teclas en el "buffer de teclado", este se vacía y en <dato> se devuelve el valor ASCII de la última tecla pulsada.

Mover_cursor(<expresión> , <expresión>)

Esta función sitúa el cursor en las coordenadas indicadas (columna 0..79, fila 0..24). El cursor permanece invisible durante toda la ejecución del programa.

Color(<expresión>)

Fija el color con el que vamos a escribir.

Aleatorio(<dato>)

Pone un valor aleatorio (indeterminado) en <dato>.

Esperar(<expresión>)

Espera el número de centésimas de segundo indicadas en la expresión.

Leer_caracter(<dato>)

Esta función devuelve en <dato> el valor ASCII del carácter que está en pantalla en las coordenadas actuales del cursor.

Poner_caracter(<expresión>)

Pone en pantalla en las coordenadas actuales el carácter cuyo ASCII indicamos, y avanza una posición el cursor.

Funciones especiales.

utilizando la sentencia 'HACER' (similar a un 'gosub').

- Un programa finaliza cuando se acaba el código (llega el final de fichero) o bien cuando, en el programa principal, llega una sentencia 'VOLVER'.

- El uso de etiquetas no esta limitado al inicio de un procedimiento: podemos utilizarlas como "identificadoras" de una parte del programa. La forma de acceder posteriormente a las partes del programa que representan estas etiquetas es mediante la utilización de la palabra reservada 'IR' seguida del nombre de la etiqueta (similar a 'goto' o 'jmp').

LAS EXPRESIONES NUMÉRICAS Y LÓGICAS

Una expresión es una fórmula que involucra constates, datos y operadores para obtener un resultado. Puede contener los siguientes tipos de elementos: constantes, referencias a datos, y operadores.

Los motivos por los que se plantea el diseño de un nuevo lenguaje son múltiples

Las constantes pueden ser de tres tipos: Numéricas; números enteros positivos o negativos (por ejemplo, '-32', '45'), de tipo lógico; una de estas dos palabras reservadas: 'CIERTO' o 'FALSO', y de tipo carácter; cualquier símbolo entre apóstrofes (por ejemplo, 'a', '=', '+'). Se debe señalar que este último tipo de constantes equivalen a escribir su código ASCII (como una constante numérica), es decir, escribir 'A' en una expresión es equivalente a escribir 65.

Dentro de una expresión se puede hacer referencia a datos o tablas de datos declaradas en el programa; por ejemplo 'contador' o 'mitabla[34]'.

Existen dos tipos de operadores: operadores numéricos, que devuelven un valor numérico entero, y operadores lógicos, que devuelven un valor lógico (CIERTO o FALSO). En el cuadro 2 aparece la lista de operadores definidos en el lenguaje Letra.

Una expresión que no involucre operadores lógicos ni constantes lógicas es una expresión numérica y por lo tanto devuelve siempre un valor numérico entero, por el contrario, una expresión que involucre constantes u operadores lógicos (aunque sólo sea uno) será una expresión lógica (como las condiciones de las sentencias 'SI...FIN', o 'MIENTRAS...FIN'). En el cuadro 3 se pueden ver algunos ejemplos de expresiones y su resultado.

LAS FUNCIONES ESPECIALES

Se trata de herramientas que permiten al programa comunicarse con "el exterior"; no pertenecen al lenguaje en sí, sino que se trata más bien de utilidades auxiliares.

En Letra no existe el concepto de librerías (como las que se utilizan en el lenguaje C), esto significa que existe un número fijo de funciones especiales predefinidas, incorporadas al lenguaje.

Todas estas funciones tienen un nombre que las identifica, debiéndose indicar a continuación de este, y entre paréntesis, los paramentos sobre los que interactúan, separando estos con

CUADRO 5

Tamaño de todos los datos	2 bytes
Longitud máxima de las líneas del fuente	254 caracteres
Longitud máxima del código generado	32 k aprox.(*)
Número máximo de etiquetas y datos	1024
Longitud del vector de nombres	12 K
Tamaño de la pila de ejecución	32 K
Espacio ocupado en pila por cada llamada	2 bytes

(*) Entendiendo que la longitud máxima del código generador engloba tanto el código como los datos.

Límites del compilador.

CUADRO 6

En el entorno de programación de Letra se muestra la información necesaria cuando se produce un error, sin embargo cuando se usa el compilador externo sólo se indica el código del error detectado, por ello se muestra a continuación la equivalencia de estos códigos con sus errores correspondientes.

01	Línea demasiado larga
02	Carácter no esperado
03	Excedida la capacidad del vector de nombres
04	Literal sin cerrar
05	Se esperaba una sentencia
06	Esta construcción no fue iniciada
07	Se esperaba un identificador
08	Se esperaba un =
09	Etiqueta inexistente
10	Esperando FIN de SI
11	Esperando FIN de MIENTRAS
12	Esperando HASTA de REPETIR
13	Se esperaba una etiqueta
14	Se esperaba]
15	Expresión incorrecta, se esperaba un valor
16	Se esperaba)
17	Se esperaba un entero
18	Se esperaba una constante
19	Identificador declarado más de una vez
20	Un dato puede contener un solo carácter
21	Demasiados valores para la tabla
22	Número fuera de rango
23	No se indicó el tamaño de la tabla
24	Dato usado a veces como una tabla
25	Se esperaba un literal de un carácter
26	Se esperaba (
27	Se esperaba un literal
28	Se esperaba un identificador de dato
29	Se esperaba una coma
30	Dato no declarado

Mensajes de error del compilador.

comas siempre y cuando se tenga más de uno. En el cuadro 4 quedan reflejadas la totalidad de las funciones especiales disponibles en Letra.

UTILIZACIÓN DEL COMPILADOR DE LETRA.

En este número, además del compilador de Letra, se entrega un editor (e.exe) que funciona a modo de entorno de programación del leguaje. Para entrar en este entorno se debe indicar como parámetro el nombre del fichero a editar o en su defecto a crear. Una vez dentro, se tiene acceso a un sistema hipertexto de ayuda sobre Letra, así como sobre el editor pulsando F1.

En Letra no existe el concepto de librerías, sino que existe un número fijo de funciones predeterminadas

Para prevenir las situaciones de bloqueo, el compilador dota a los programas ejecutables de un mecanismo de recuperación del control que consiste en abortar la ejecución de cualquier programa escrito en Letra en el momento en el que se pulsa la tecla ESC. No obstante, no existe ningún mecanismo por el que se pueda determinar donde se quedó bloqueado el programa, debiéndose recurrir a la inserción de trazas, si no se encuentra el origen del bloqueo.

LIMITACIONES TÉCNICAS DEL COMPILADOR

A parte de las limitaciones inherentes al lenguaje, existen las limitaciones del compilador. Este genera a partir de los programas fuente, programas ejecutables (.COM) para el entorno MS-DOS. La ejecución de los programas se realiza siempre en modo texto estándar. En el cuadro 5 aparecen reflejadas las limitaciones técnicas del compilador de Letra que incluimos con la revista.

No existen otras limitaciones significativas del compilador, ni en tamaño de los identificadores, ni en número de anidamientos de las sentencias, ni en longitud de las expresiones, etc.

OTROS EJEMPLOS MÁS COMPLEJOS

En el disco de la revista se hace entrega del entorno de programación Letra, así como de algunos ejemplos. Dichos ejemplos explotan al completo las posibilidades del lenguaje, por lo que de su examen se puede corroborar todo lo que aquí se explica.

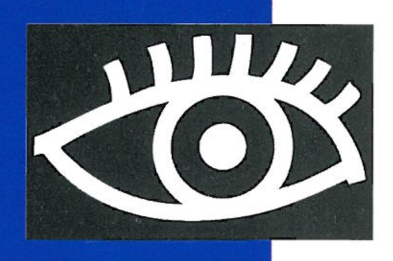
PRÓXIMO NÚMERO

Hasta aquí se ha descrito suficiente información de Letra como para poder comenzar en el próximo número a programar un compilador para él. Hasta entonces se puede utilizar una versión ejecutable ya finalizada de este compilador, que nuevamente se entrega en el disco de la revista, ya que para cuando se esté programando el compilador no puede haber ninguna duda sobre el funcionamiento de este sencillo lenguaje.

Estamos a la espera de la llegada de sus sugerencias sobre el compilador para siguientes artículos, y también pueden mandarnos los programas que implementen en Letra. Estos serán publicados en próximos números de la revista, siempre que reúnan un mínimo de calidad.

LA GRAN PARTY ESPAÑOLA

José María Álvarez



Como ya se anunció en el artículo dedicado al Assembly en el pasado número de Octubre, se ha celebrado la Euskal Party III coincidiendo con el puente del Pilar, los días 12, 13, 14 y 15 de Octubre. Dicha reunión (denominada "party" entre los entendidos) tuvo lugar en la acogedora ciudad de Tolosa, a pocos kilómetros de San Sebastián.

LA EUSKAL PARTY III

La party de este año ha sido para organizadores y asistentes algo especial. En primer lugar, era la primera vez que el concurso tenía categorías dedicadas al PC, como por ejemplo el concurso de música multicanal. Esto hizo incrementar el número de asistentes al certamen (320 en total, divididos en 145 personas inscritas en el área de PC, 156 en el área de Amiga y 19 en otras áreas). Por otra parte, algunos elementos técnicos de los que se han dispuesto para esta party superan, incluso, a los del ya mencionado Assembly, la party por excelencia. Por ejemplo, los participantes y el público asistente pudieron ver las demos en una pantalla gigante de 6x4 metros, la más grande utilizada en un evento de este tipo. En fin, como suele decirse las cosas cambian, y la organización de la Euskal Party III ha demostrado que no hay que ir a remotos parajes de Europa para hacer las cosas bien.

EL LUGAR DE LA PARTY

Si el Assembly 95 se celebró en un pabellón de una feria de muestras, la Euskal Party III tuvo lugar en un pabellón polideportivo. Los participantes y

La Euskal Party ha celebrado este año su tercera edición, la cual ha destacado por su organización y unos brillantes aspectos técnicos. El índice de participación ha sido alto, siendo éste el primer año en el que el concurso contaba con categorías dedicadas al PC.



asistentes en general traían sus equipos (PCs, Amigas, algún PowerPC y PC RISC, equipos de música, etc...) los cuales conectaban a una red de enchufes proporcionada por la organización, a lo largo del citado pabellón. La propia organización dispuso de un Amiga 4000 y de un PC 486 DX2 a 66 Mhz para visualizar las demos.

Debajo del marcador electrónico se situaron dos pantallas gigantes, una para la visualización de las creaciones preparadas para la party, y otra más pequeña para realizar las presentaciones. Justo enfrente, al otro lado del pabellón, estaban las gradas, las cuales acogieron a gran número de espectadores a lo largo de los 4 días que duró el evento. Para hacer más espectacular la muestra de producciones se apagaban las luces del pabellón, se subía el volumen del equipo de megafonía instalado, y se mostraban producciones antiguas durante los momentos de espera.

La parte baja del pabellón (duchas y vestuarios) fue habilitada para que la gente pudiera colocar sus sacos de dormir, aprovechando las pocas horas de sueño de las que se dispuso durante todo el evento. Vale la pena comentar que hubo asistentes que sumidos en el ambiente de la party estuvieron 3 días sin dormir.

Otro aspecto a agradecer fue la disposición en la entrada del recinto de unas barras donde se servían bebidas, bocadillos y aperitivos a un precio módico, y a cualquier hora del día (y de la madrugada). Al parecer sirvió de ayuda para la financiación del viaje de fin de curso de unos estudiantes de COU.

EL AMBIENTE DE UNA PARTY

Los espectadores asistentes se quedaron asombrados de aquel cúmulo de cables, ordenadores, equipos de música, programadores, incluso cafeteras y hornillos de gas. Abajo, en la pista del pabellón, muchos de los participantes que presentaban algún trabajo a concurso aprovechaban las últimas horas que disponían antes de la entrega (la denominada "deadline"), para terminar su trabajo. Incluso hubo un grupo que adoptó como nombre dicha palabra para confeccionar una demo en el preciso día de la entrega, quedando clasificados finalmente en el tercer puesto. Como afirmaba el grupo Iguana en su demo presentada (a la postre la ganadora de la categoría) la gracia de participar en un evento de este tipo es acabar la creación en el último momento (la frase exacta que citaban era party coding rules). A pesar de eso hay grupos que no consiguen acabar a tiempo, o se quedan sin terminar pequeños detalles (como el típico scroller con saludos a otros compañeros de fatiga). No obstante, los trabajos presentados fueran de tiempo, a pesar de no entrar en concurso, fueron mostrados al público en la pantalla gigante.

Una muestra de que el certamen gana en popularidad fue la presencia de periodistas y cámaras de televisión de cadenas autonómicas (como TV3, televisión de Cataluña). Además, y en colaboración con los organizadores, se

Había, por primera vez, categorías dedicadas al PC

DEMOS DE PC

Lugar	Puntos	Título	Grupo/autor
1	715	PUMP	Iguana
2	595	Inpro VICE	TLOTB
3	280	Fistropía	Dead Line

DEMOS DE AMIGA

Lugar	Puntos	Título	Grupo/autor
1	1380	El ocaso del Payaso	Goblins
2	550	Non	Ozone
3	515	Short	Capsule

INTROS DE PC (64Kb)

Lugar	Puntos	Título	Grupo/autor
1	795	Psicho Therapy	TLOTB
2	765	Romantic	Miracle
3	355	Wintermute	Gore Design

INTROS DE AMIGA (64Kb)

Lugar	Puntos	Título	Grupo/autor
1	1085	Career	Capsule
2	835	Novato	Asier Lasa

INTROS DE PC (64Kb)

Lugar	Puntos	Título	Grupo/autor
1	655	Magnetic Fields 6	D.Vader / The Banner
2	625	Sky Demo	Marconi
3	555	Eusk 4	Cranky / TLOTB

INTROS DE AMIGA (4Kb)

Lugar	Puntos	Título	Grupo/autor
1	1260	D.Miguel de Cervan	Phornee / Goblins
2	660	Oxygene IV	Sapphire / Centolos
3	360	El toro de Okam	Birra / Goblins



instaló en el recinto un stand de Radio Popular, en el que diariamente se celebraban charlas de temas como por ejemplo "RISC versus CISC", el hardware del PC y del Amiga, y el estado de "la escena" en España (así es como denominan los programadores a este mundo de gráficos y sonido).

LOS CONCURSOS

Los concursos presentados en la edición de la Euskal Party de 1995 se dividieron en demos, gráficos y sonido. Las categorías en el concurso de demos fueron las de intros (producciones con un tamaño máximo de 64Kb), 4KB intros (de ese mismo tamaño) y demos (también denominadas mega-demos debido a su gran tamaño). El concurso de gráficos contó con las categorías de gráficos 2D (dibujos hechos "a mano" con un programa de dibujo), gráficos 3D (escenas renderizadas por un programa de 3D), y fast-2D (dibujos hechos durante la party). El concurso de música se dividió en canciones de 4 canales, canciones de hasta 32 canales (multicanal, reservado sólo para PC), y fast-music (canciones realizadas en un tiempo determinado sobre un tema que elige la organización. En este caso las canciones versaron sobre el mundo de la televisión). Estas canciones podían ser módulos de sonido de cualquier tipo (MOD, S3M, XM, MID, etc...).

Un peculiar concurso añadido a éstos fue el de lanzamiento de disquete, en el que los participantes compiten por saber quien es capaz de lanzar más lejos un disquete de 3 1/2. El ganador logró rebasar los 40 metros de distancia.

Al contrario que en el Assembly de Finlandia, en la Euskal no hubo preselección en las producciones presentadas, debido al menor número de éstas en comparación con el Assembly. Al fin y al cabo la Euskal todavía tiene carácter de certamen nacional.

LAS PRODUCCIONES DE ESTE AÑO

Los ganadores de cada categoría se deciden por votación popular entre los asistentes a la party. Para ello los organizadores habían elaborado un programa que permitía elegir las 3 mejores creaciones en cada apartado, el cual distribuyeron entre los asistentes. Los puntos concedidos eran 15 para la 1ª elección, 10 para la 2ª y 5 para la 3ª.

La clasificación de los 3 mejores en cada categoría, junto con el número de votos obtenidos, puede verse en la siguiente tabla:

Este año la demo ganadora ha sido "Pump", del ya famoso grupo Iguana (recordemos que el año pasado quedó en tercer lugar en el Assembly), los cuales habían reservado sus efectos de última creación para esta Euskal Party. Como nos tienen acostumbrados, asombraron a todos los presentes.

Como ya se comentó en el anterior artículo sobre el Assembly, la técnica más utilizada en las demos presentadas a concurso este año ha sido la de Phong, la cual proporciona a los objetos creados mediante estructuras en tres dimensiones una textura de brillos como si dicho objeto fuera de plástico o metal. Tampoco han faltado unos objetos toroidales redondeados mediante la técnica de Phong, conocidos comunmente como donuts. También han proliferado efectos como caleidoscopios, plasmas o túneles (con texturas aplicadas).

CONCLUSIONES

Para concluir este reportaje, y para aquellos que nunca han asistido a una party, hay que destacar que es algo espectacular. Los espectadores asistentes quedaron impresionados con las posibilidades gráficas y sonoras de los trabajos presentados, observando qué es lo que se puede llegar a hacer con un PC (o con un Amiga). Entre los participantes se respiraba un aire "especial", que les incitaba a trabajar sin descanso en demos, canciones, imágenes, etc. sin más codicia que el hecho de ser reconocido públicamente como el ganador del concurso.

Una party es una buena oportunidad para ver lo último en técnicas de gráficos y sonido, los programadores (conocidos comúnmente como coders en el ámbito de la "escena") intercambian conocimientos, adquiridos mediante rutinas que a veces implementan complejas fórmulas de física y matemáticas (recordemos, por ejemplo, la famosa geometría fractal).

La organización prácticamente no falló en nada. Quizá hubo poco público en la entrega de premios, algo achacable al cansancio general, que hizo que muchos se fueran el último día de buena mañana, antes de la entrega. Los premios, eso sí, fueron entregados inmediatamente de manos del mismísimo alcalde de la ciudad de Tolosa.

Una de las pocas cosas que se echaron en falta respecto del Assembly fue la falta de una red para conectar los ordenadores, y poder así disfrutar de juegos en red o mantener conversaciones con cualquier otro usuario. Como se suele decir, tiempo

GRÁFICOS 2-D			
Puntos	Título	Grupo/autor	
1	930	Gumball HiRes 16 C.	GOD /LLFB
2	815	La Familia Addams	Phornee / Goblins
3	375	Split the Time	Yacare / Zoran

GRÁFICOS 2-D, CONCURSO RÁPIDO (FAST 2D)			
Lugar	Puntos	Título	Grupo/autor
1	655	Froni	Phornee / Goblins
2	555	Machine	Ripp / Please Wait
3	465	———	Yacare / Zoran

GRÁFICOS 3-D (RAYTRACE)			
Lugar	Puntos	Título	Grupo/autor
1	715	Amiga Live	Zen / Nexus
2	420	Casa Harrison	Asier Hernaez
3	260	Bath	Jose Menendez

al tiempo. Lo visto en esta party demuestra que ganas de trabajar no faltan, y como los propios asistentes reconocían, la mejora de nivel respecto a las anteriores ediciones ha sido espectacular. Algo que sí que hubo, y mucho, fue cortesía y buen trato por parte de la organización, la cual accedió de buen grado a la prueba en la pantalla gigante de versiones previas de programas, que incluso pedían a los presentes. Hubo paciencia por su parte en la visualización de demos que inicialmente fallaban, y el trato era en todo momento personal y nada pedante, a diferencia de otros eventos de este tipo. Una vez más, un diez por su parte.

En resumen, es algo que vale la pena no perdérselo, además teniendo en cuenta que se realiza en nuestro propio país. La fama y prestigio de esta party depende del apoyo que entre todos podamos darle, y espero que estas palabras sirvan de ayuda para futuras ediciones.

MÚSICA (MÓDULOS DE 4 CANALES)			
Lugar	Puntos	Título	Grupo/autor
1	505	Euphonix	Estrayk / Capsule
2	505	Something Good	Evelred / Capsule
3	325	Afrika 2	Samplemin / GenX

MÚSICA MULTI-CANAL (MÓDULOS DE HASTA 32 CANALES)			
Lugar	Puntos	Título	Grupo/autor
1	425	The Voice	Evelred / Capsule
2	355	Sing	Dem / Zoran
3	295	Lonely Fall	Thander/CrystalShade

MÚSICA, CONCURSO RÁPIDO (FAST MUSIC)			
Lugar	Puntos	Título	Grupo/autor
1	465	Pequeño Pony	R. Rodriguez / Serpix
2	450	D Can PC	Shade / Alien
3	425	Campeones	M. Recondo / Serpix

Update now!

WATCOM

Compiladores para lenguajes profesionales
- Lenguaje C/C++V.10.0 CD ROM
- Lenguaje Fortran 77/32 Bits v 9.5

Ambiente de desarrollo visual (Front End)
- Lenguaje OS/2 Rexx
- VX-REXX v.2.1 Standard Edition
- VX-REXX v.2.1 Client/Server

Bases de datos relacionales (llamada SQL) v 4.0
- Plataformas: Windows, NT, OS/2, DOS y Netware
- Usuarios: Standalone - Network Server
- Lenguaje: C/C++

RAIMA

Raima Database Manager: Base de Datos
- Lenguaje: C
- Sistema Operativo: DOS, Windows, OS/2, UNIX System V, Berkeley 4.2 AIX, Sun OS, SCO, QNX, YULTRIX y VMS
- Plataformas: DOS, OS/2
- Modalidades: Module y System
- Royalty Free

VELOCIS

Velocis: Base de datos. Tecnología Cliente/Servidor
- Lenguaje C/C++
- Sistema Operativo: Windows, NT y UNIX
- Plataformas: Windows, NT, OS/2, Netware 386 NLM, OS/2 y UNIX
- Velocidad de acceso 1/3 más rápido que otras B/Datos

PHARLAP

Gestiona la memoria optimizando los límites de todas las versiones DOS

SEQUITER
Software Inc.

Librerías de programas para trasladar aplicaciones de Dbase a C/C++
- CodeServer v.5.1 Cliente/Servidor

OFERTA ESPECIAL

Para los lectores de
SÓLO PROGRAMADORES

Infórmate llamando:
Tel.: (93) 225 39 95
Fax: (93) 225 40 0[illegible]

CMR
Competitive
Manufacturing
Research

C. Lope de Vega, 21 bjs.
08005 Barcelona

EL SIGUIENTE PASO

David Aparicio

En una competencia feroz por dominar el mercado, Intel prepara los últimos detalles del lanzamiento de su último logro tecnológico: El microprocesador P6, recientemente bautizado como Pentium Pro.

Desde 1974, año en el que Intel lanzó el 8080, el mundo de la informática ha conocido la familia de microprocesadores más popular (y controvertida) del mercado. Gracias a una arquitectura abierta, y a la suerte y el genio comercial del actual gigante del software, Microsoft, la familia Intel x86 ha dominado desde el principio el mercado de los ordenadores de sobremesa, por delante de su más fiero enemigo: Motorola. En un entorno más y más competitivo, las necesidades de crear una máquina multitarea, llevaron a Intel en 1985 al lanzamiento del 386, la base de la arquitectura que ahora hereda el P6.

A partir del Pentium, Intel introduce nuevos conceptos, heredados del mundo RISC, para mejorar radicalmente el rendimiento de sus microprocesadores. ¿Cómo continuará el proceso?

LA COMPATIBILIDAD

A mediados de septiembre, Intel se pronunció sobre el nombre de su nuevo procesador: El P6 fue bautizado como *Pentium Pro*. Se trata de una arquitectura cuyo objetivo es mejorar el rendimiento del Pentium 100 Mhz, pero utilizando su mismo proceso de fabricación. La polémica decisión de mantener el nombre de la CPU que se vio involucrada en el famoso bug de la división parece ser una apuesta publicitaria de la superación sin traumas de aquel problema, pero también el velado mensaje de que este micro no representa una nueva línea de producto, sino una continuidad del mismo concepto.

Para conseguir el aumento de rendimiento, Intel se ha basado en conceptos RISC, para aplicarlos al corazón de un cuerpo todavía CISC. Conservando el mismo juego de instrucciones, este micro incorpora arquitectura superescalar, con un *pipeline* de 14 etapas, ejecución especulativa, y un aumento del caché L1 resultando todo ello en un encapsulado en una sola pastilla de una CPU de 5,5 millones de transistores y una caché L2 de 15,5 millones de transistores. Para hacernos una idea, el Pentium dispone de 3,3 millones mientras que un PowerPC 604 tiene uno de 3,6 millones.

La integración del cerebro y la caché en un sólo encapsulado (observe la figura 1) pretende simplificar el proceso de diseño de placas base para esta arquitectura. Intel desea una rápida incorporación de las nuevas máquinas, dado que tanto Cyrix como AMD están a punto de lanzar los competidores del Pentium, y el PowerPC amenaza con introducirse en el mercado de compatibles PC.

Con el diseño mejorado que incorpora, Intel estima que este micro pasará de una velocidad inicial de 132 Mhz hasta los 250 Mhz cuando el proceso de fabricación madure, manteniendo el liderazgo frente a la competencia mientras el mercado de software se decide a proporcionar una base sólida para un cambio radical en el diseño de microprocesadores.

CONCEPTOS

El Pentium introdujo varios conceptos nuevos sobre el 486. La primera, de carácter comercial, donde su nombre dejaba de ser un número para impedir que la competencia aprovechara la imagen de los productos Intel en su propio provecho. Por otra parte, un bus de datos de 64 bits permite mejorar el comportamiento en proceso de punto flotante.

El sistema de caché, separando código y datos, otorgó la capacidad de pre-

decir la dirección de salto en el flujo del programa (*ejecución especulativa*), teniendo listas las instrucciones adecuadas para mantener un aporte efectivo de código para la arquitectura *superescalar*.

Este último concepto significa que el microprocesador está ejecutando varias instrucciones en un mismo ciclo de reloj. Por ejemplo, mientras se esta accediendo a memoria para recoger los datos indicados por una instrucción, también se puede estar decodificando el operando de la siguiente.

El *pipelining* se refiere a la capacidad arquitectural de descomponer cada instrucción en varios pasos, de forma que el micro sea capaz de ejecutar un aspecto de una instrucción al mismo tiempo que otro aspecto de otra.

Todos estos avances resultan en una mejora apreciable de rendimiento frente al Pentium, puesto que el Pentium Pro explota estas características más profundamente. Su arquitectura superescalar admite 14 etapas, con lo que se puede simultanear la ejecución de más instrucciones. El sistema de ejecución especulativa puede predecir entre 10 y 15 saltos anidados, con probabilidad de acertar en un 90 por cien de los casos. Pero lo que resulta innovador, es la ejecución fuera de secuencia de las instrucciones. Hasta el Pentium, éstas se ejecutan en el orden indicado en el programa. En este nuevo micro, se pueden ejecutar instrucciones sin seguirlo necesariamente. Observe esta secuencia de código:

A) r1 = mem(r0)
B) r2 = r2 + r1
C) r3 = r3 + 1
D) r4 = r6 - r5

Mientras se está produciendo un acceso a memoria (A), un Pentium debe permanecer esperando. Pero el P6 sigue tomando instrucciones, y decide si dependen de la que está bloqueada. En el ejemplo, (B) debe esperar el resultado de (A). Por el contrario (C) y (D), pueden ser ejecutadas, guardando después los resultados intermedios de los registros. Para cuando el acceso de memoria está completado, parte del programa que sigue ya está listo, con lo que se simplemente se validan los resultados, produciendo una gran ganancia en rendimiento. Esto se denomina ejecución "out-of-order", y la denominaremos, para entendernos, *ejecución desacoplada*.

LOS DETALLES DE LA ARQUITECTURA

Pasemos a analizar la arquitectura del nuevo microprocesador con mayor detalle. Observe la figura 2, donde se muestran los componentes básicos del P6.

A grandes rasgos, el Pentium Pro dispone de un caché L2 integrado en el propio encapsulado, que optimiza el acceso a la memoria del sistema, con un ancho de 64 bits. El alto número de transistores que integran este caché, unos 15,5 millones, se debe a que se trata de una memoria estática con seis transistores por bit., de forma que no necesita refresco periódico y su tiempo de acceso es ínfimo.

El caché L1 está separado en dos partes independientes, una para código y otra para datos.

El caché de código contiene instrucciones en las direcciones indicadas por el IP a leer. El BTB trata de predecir la dirección de los saltos y pide nuevas zonas de programa al caché, produciendo así la ejecución especulativa.

El decodificador recoge los códigos de los operandos y los traduce al micro-código del P6, que es ajustado por el MIS para rellenar el pipelining y explotar la característica superescalar. Debido a la ejecución especulativa y desacoplada, el P6 dispone de un número de registros internos mucho mayor del que puede ver el software, siendo 40 registros físicos frente a 14 arquitecturales.

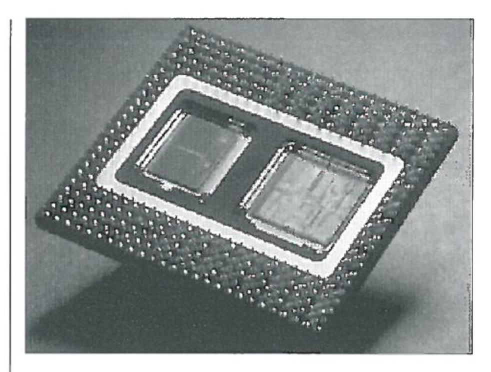


Figura 1: El Pentium Pro: Detalle del encapsulado.

Conforme el Pentium Pro especula sobre los resultados de la ejecución, asocia los registros software con sus propios registros internos, mediante la RAT, y anota una transacción en el buffer de reajuste (ROB). Si la ejecución se produce realmente como el micro ha previsto, el módulo de reservas recogerá los resultados (RRF) y hará efectivas las operaciones sobre la memoria (MIU). Si, por el contrario, la ejecución especulativa ha fallado, dicho módulo se encarga de invalidar las transacciones asociadas, vaciar el pipeline y ejecutar en la zona de programa correcta.

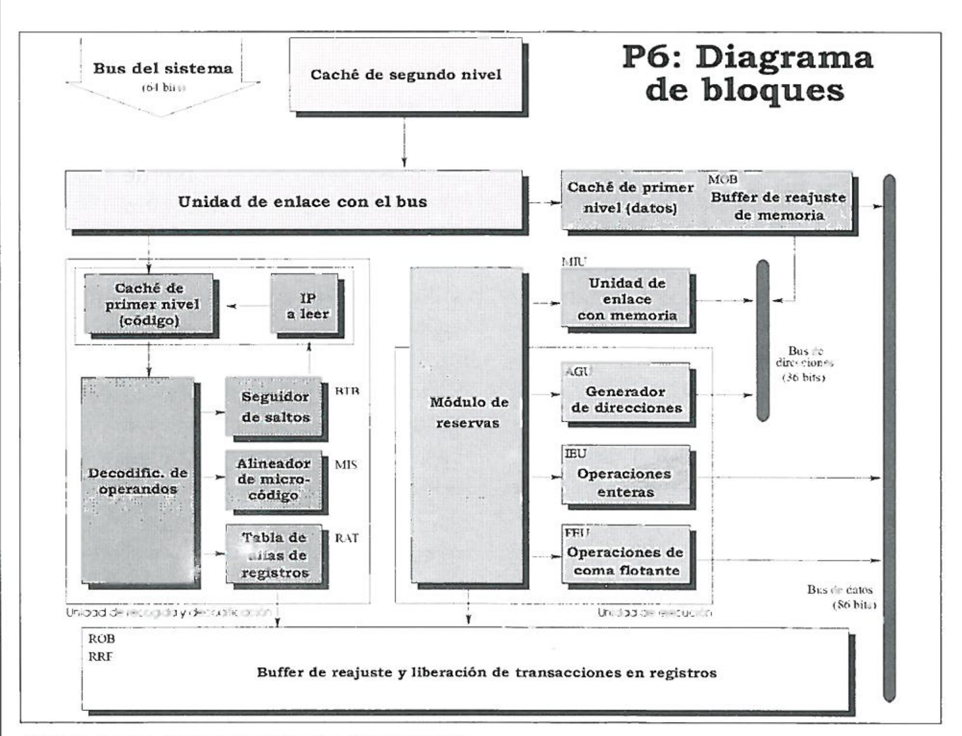


Figura 2: Los componentes del Pentium Pro.

En la unidad de ejecución podemos observar que existen varios integrantes: el AGU, IEU y FEU. Estos permiten simultanear instrucciones sobre enteros (hasta dos), direcciones (lectura y escritura) y punto flotante, para hacer efectiva la ejecución de varias operaciones en cada ciclo.

Finalmente, el flujo de datos que pasa por el caché L1 correspondiente cuando se requiere debido al flujo del programa, se reordena en el MOB para hacer que se corresponda el flujo desacoplado del micro y la secuencia que indica el código. El control sobre este aspecto lo realiza el módulo de reservas, a través de la unidad de enlace (MIU).

El motor de la ejecución desacoplada (no secuencial) es el módulo de reservas, que se encarga de averiguar las transacciones que pueden progresar en la ejecución, en base a:

- que no dependan de instrucciones previas
- que tengan disponibles los recursos que necesitan

Cuando las transacciones se completan, el módulo de reservas prosigue la ejecución de otras, esperando liberar a las primeras cuando el flujo del programa se cumpla.

Como hemos comentado anteriormente, el uso de código que mezcla registros de 8, 16 y 32 bits ocupa mayor número de registros internos, hace mas dependiente unas instrucciones de otras, y da menor libertad al módulo de reservas, con lo que se dificulta el rendimiento del microprocesador.

LLEGAN LOS INCONVENIENTES

Sin embargo, en pruebas preliminares, se está comprobando que el rendimiento del Pentium Pro no es tan radical como era de suponer. Si pensamos que los prototipos probados por Intel están especialmente equilibrados, y que los sistemas que se lancen al mercado no tendrán un diseño global tan bueno, resulta que las medidas publicadas hasta ahora son las mejores posibles para 132 Mhz. Sorprendentemente, los resultados se equiparan al Pentium de 132 Mhz, y se encuentran por debajo del PowerPC 604, su competidor más directo.

En primer lugar, el concepto de optimización ha variado. Si un trozo de código utiliza repetitivamente el mismo registro para almacenar resultados intermedios, la capacidad de ejecutar de forma desacoplada disminuye, aunque el programador piense que "ahorra registros". Por otro lado, la compatibilidad con la familia 8086 proporciona un número de registros insuficiente, si se compara con arquitecturas RISC, y dificulta la ejecución eficiente en una arquitectura superescalar.

El diseño del código que generan los compiladores está íntimamente ligado al rendimiento de los procesadores RISC. Debido a la aproximación de la arquitectura Intel a este comportamiento, el código resultante de viejos compiladores no puede aprovechar adecuadamente la nueva arquitectura.

Por otro lado, cuanto mayor es el tamaño del pipeline, mayor penalización se produce cuando el micro se equivoca al predecir una bifurcación en el programa. Se debe deshacer el resultado de todas las instrucciones completadas, realizar nuevos accesos a memoria y comenzar de nuevo. Intel también ha introducido en el P6 hardware específico para aliviar esta circunstancia.

Otro problema que penaliza al nuevo micro es que las pruebas se realizan en muchos casos sobre código de 16 bits. Observe el siguiente código:

```
MOV  AX,(DI)
XOR  AH,AH
AND  EAX,EBX
PUSH AL
```

En todo momento, se están empleando operaciones sobre el mismo registro, en combinaciones de 8, 16 y 32 bits. Esto choca con el concepto del P6, y lo ralentiza de forma apreciable. De hecho, el Pentium Pro actual ejecuta código de DOS/Windows 3.1 más lentamente que un Pentium a igual frecuencia de reloj. Recordemos que ciertos trozos de código de Windows 95 siguen siendo de 16 bits, y que el usuario puede verse tentado (u obligado por su bolsillo) a ejecutar aplicaciones que no han sido recompiladas para el nuevo entorno. Todo esto penaliza al nuevo micro, y le hace perder terreno frente a sus competidores

PRUEBAS DE RENDIMIENTO

En primer lugar, hemos de puntualizar que estos resultados se han realizado en base a publicaciones por parte del fabricante, puesto que, en el tiempo de redacción del artículo, Intel dispone de sistemas para depuración propia, pero no ha empezado a distribuirlos para evaluación. La figura 3 compara el progreso de la familia Intel, tomando como factor su propia unidad de medida, la cual mezcla pruebas sobre operaciones enteras y de punto flotante, en 16 y 32 bits. Exactamente, el índice iCOMP se basa en los siguientes aspectos:

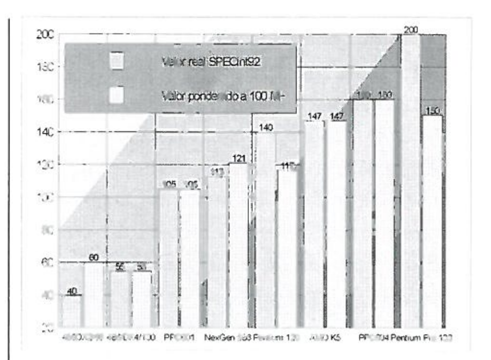


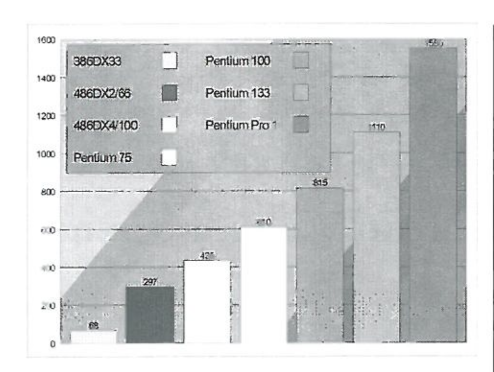
Figura 4: Rendimiento SPECint92.

- 25% operaciones enteras en CPU (SPECint92)
- 5% operaciones de punto flotante en CPU (SPECfp92)
- 68% operaciones en aplicaciones DOS (Ziff-Davis PC Bench)
- 2% operaciones de punto flotante (Whetstones)

En estos momentos, el iCOMP para el P6 no se encuentra disponible, así que lo hemos ponderado en base a los valores SPECint (70%) y SPECfp (30%), con respecto a valores de referencia de la familia Pentium. Se podría aventurar que el valor real sería similar o inferior al que se proporciona en la figura, pero hemos de recordar que especulamos sobre un sistema que no hemos podido probar antes de la redacción del artículo.

Las figuras 4 y 5 muestran el rendimiento puro de varios microprocesadores en operaciones enteras de 32 bits (SPECint92) y de punto flotante (SPECfp92). Hemos representado los resultados en valor absoluto (cada micro a su frecuencia de reloj), y también ajustando las pruebas en base a una frecuencia fija, de 100 Mhz, para tratar de representar el verdadero rendimiento interno de cada arquitectura.

Como podemos observar, el rendimiento ponderado en el 486 parece dis-


Figura 3: Rendimiento iCOMP.

minuir cuando se pasa de duplicar a triplicar el reloj. Por otra parte, hay una mejora apreciable en ambos tests a partir del Pentium, resultado de introducir una arquitectura superescalar, y un bus de datos de 64 bits. Hemos querido comparar los resultados con los de los competidores directos de Intel. Los procesadores PowerPC parecen obtener mejores resultados ponderados, lo cual parece señalar que tiene mayor facilidad para aumentar el reloj del sistema y mejorar el rendimiento.

El actual proyecto de AMD, el K5, con un diseño muy similar al del Pentium Pro, y que aparecerá en nuestras mesas casi al mismo tiempo que éste, se encuentra a un nivel comparable con el micro de Intel. El procesador de NexGen, sin embargo, incorpora una tecnología a medio camino entre el P5 y el P6, y parece dedicado a competir al nivel del Pentium, quedando descolgado de la nueva generación de micros.

Teniendo en cuenta que el 486 está próximo a ser descatalogado por parte de Intel, no parece probable que aparezca a mas de 100 Mhz, y, por otro lado, tal vez no ofrezca un rendimiento efectivo a mucha mayor velocidad. Sin embargo, AMD acaba de lanzar dos versiones, a 120 y 133 Mhz, que se pueden equiparar a un Pentium 75 por un precio realmente competitivo.

Por otro lado, el proceso de fabricación de Intel, heredado del P5, de 0,6 micras, acaba de pasar a 0,35 micras (con un prototipo a 200 Mhz), lo que se traduce en una mejora considerable en el rendimiento. El Pentium todavía puede llegar a alcanzar entre 150 y 180 Mhz, mientras que se especula que el Pentium Pro sería capaz , con el nuevo proceso, de alcanzar los 250 Mhz (suponiendo un comportamiento lineal, esto resultaría en un SPECint de cerca de 380). Por el momento, el punto de partida para el P6 y el PPC604 está en 132 Mhz, y sólo el límite del reloj al que se pueda someter a cada arquitectura marcará la mejor opción.

CONCLUSIONES

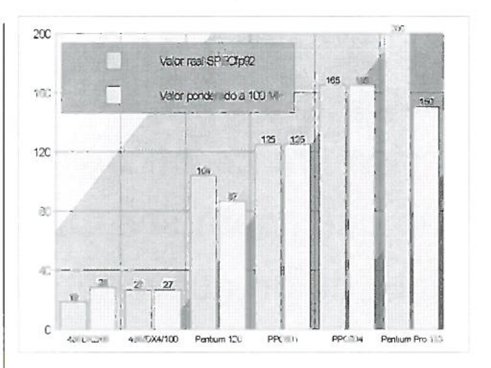
Parte de la estrategia de dominio del mercado que Intel y Microsoft han mantenido en casi veinte años es también su principal debilidad. Debido a la inercia en el software, Intel se ha visto obligado a mantener una compatibilidad binaria que perjudica el rendimiento de su arquitectura. El nuevo P6 todavía soporta un sistema operativo de 16 bits y monotarea porque Microsoft se encuentra ahora en una etapa de transición a 32 bits que debió comenzar hace tiempo.

El futuro Cairo (y el clásico Unix) marcarán, en un plazo de cuatro años, una etapa unificadora. Hasta entonces, Windows 95 es el encargado de hacer de puente entre el caduco DOS y los 32 bits. Por el camino, perderemos aplicaciones, programas y juegos que recordaremos con nostalgia (Todavía muchos recordamos los increíbles juegos de 8 bits). Pero la senda que habrá de recorrer el software en este tiempo, es otra historia.

El Pentium Pro se beneficiará de un compilador de 32 bits que "comprenda" los aspectos que puedan resaltar las ventajas del nuevo micro. Como en el mundo RISC real, el compilador es una baza importante. Por otro lado, Intel afirma que, con el cambio de tecnología de 0,6 a 0,35 micras, el rendimiento del P6 superará al Pentium en todos los aspectos, incluso en 16 bits.

Al usuario final también le interesará saber el precio de salida del nuevo procesador. En principio, la integración del caché L2 en el encapsulado, aumentará el precio del chip. Sin embargo, hay que tener en cuenta que los otros sistemas también utilizan un caché externo que se paga al adquirir la placa base, y cuya dificultad técnica ralentiza la incorporación de nuevos fabricantes. En este aspecto, recuerde que sigue habiendo misteriosamente pocos sistemas PowerPC.

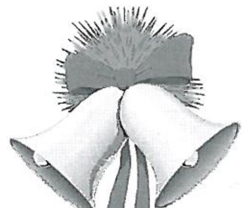
Por último, el techo de rendimiento para esta generación de micros no está claro, pero podría ser menor para el Pentium Pro que para PowerPC debido al mayor número de transistores que integra el primero.


Figura 5: Rendimiento SPECfp92.

Además, el factor de la pérdida de rendimiento del Pentium Pro en aplicaciones 16 bits puede producir curiosos efectos en el mercado. Si Microsoft no consigue enterrar el mundo DOS y obligar a las empresas de software a producir código radicalmente nuevo, Intel se verá obligada a depender del éxito de otros sistemas operativos de 32 bits multitarea (como el robusto OS/2, si alguna vez supera el trauma de los *drivers*, o el clónico Unix que crece con mayor rapidez: Linux) para la propia supervivencia del Pentium Pro, mientras mejora su propio proceso de fabricación. A su vez, el posible éxito de otros sistemas operativos podría frenar los planes de Bill Gates para mantener el monopolio de su sistema operativo en el mundo del PC, y le obligaría a ampliar clientes siguiendo un desarrollo multiplataforma, como empieza a verse en NT. De nuevo, ambos gigantes de la informática se asegurarían mutuamente el futuro trabajando en equipo.

El Pentium Pro representa un esfuerzo tecnológico apreciable por parte de Intel. Junto al factor de la política de precios que se mantenga, habrá que ver el impacto sobre el software del mercado en los próximos dos años. No olvidemos que el P7 está ya en fase de desarrollo en el cuartel general de Intel, en conjunto con HP, para intentar unificar la arquitectura PA-RISC y la compatibilidad Intel x86 e intentar un primer asalto al mundo de las estaciones de trabajo.

De todos estos cambios, el usuario final será el beneficiario. Parece que el año 1996 nos traerá emociones fuertes a todos.



SÓLO PROGRAMADORES UN AÑO MÁS

Diciembre es un mes especial para todos, ideal para hacer resumen del año que agota sus últimos días, y también para plantearse el futuro inmediato de un nuevo año entrante.

Desde estas líneas la redacción de Sólo Programadores quiere desear una feliz Navidad y un próspero Año Nuevo a todos los lectores de esta revista. Aprovechando esta ocasión queremos comentar algunos aspectos sobre la actual situación de "Solop" y los planes futuros para ella. Lo primero decir que Sólo Programadores goza de buena salud, el nivel de ventas es bueno para una revista de carácter eminentemente técnico, pero más importante para nosotros es el prestigio que posee entre los profesionales y estudiantes del campo de la informática y especialmente la programación. De hecho son ellos los que realizan la revista y seguro que todos vosotros en cualquier momento podéis entrar en ella. Nosotros, por supuesto, no nos conformamos con lo hasta ahora conseguido, y queremos seguir mejorando Sólo Programadores número a número. Cómo conseguirlo no es fácil, pero un primer paso es ampliar nuestro campo de colaboración con los mejores programadores del sector. De hecho lo estamos logrando, y próximamente comenzaran a firmar artículos algunos profesores provenientes de los claustros de las mejores universidades españolas, así como grandes profesionales del sector privado que gozan de una gran reputación debido a sus trabajos. Por supuesto que continuarán la mayoría de nuestros actuales colaboradores, a los cuales desde estas líneas queremos dar las gracias por su brillante trabajo y desearles especialmente a ellos una feliz Navidad y próspero 1996; entregar artículos a tiempo, frecuentemente les cuesta perder horas de sueño y algún que otro fin de semana. Un dato anecdótico es que el 100% de nuestros colaboradores son del sexo masculino y seguro que hay muy buenas programadoras con gran potencial para escribir buenos artículos, tenéis que animaros y uniros a la familia de Sólo Programadores.

Un segundo paso para mejorar la revista es regalar más CD´s y vamos a intentar que si no en todos los números, en la mayoría de ellos podamos entregar un CD atractivo. Aprovechamos la ocasión también agradecer a tantas personas que desinteresadamente nos mandan sus programas para que los incluyamos en nuestro CD-ROM y también los que nos mandan esbozos de artículos, guiones e ideas y que algunos finalmente pasan a ser firmas habituales de Sólo Programadores.

Un tercer paso que deseamos conseguir es llegar a todos los rincones de España puntualmente, si bien sabemos que en muchos sitios no hay problema existen algunas localidades donde realmente es difícil conseguir un ejemplar de Sólo Programadores. Este esfuerzo también lo queremos realizar para que la revista llegue lo antes posible al otro lado del Atlántico, desde donde recibimos muchas cartas, especialmente de Argentina.

Resumiendo un poco lo que ha sido este año en Sólo Programadores citar nuestra continua atención al lenguaje C++, al Ensamblador, al Clipper y en general el campo de las bases de datos, al mundo de la programación bajo Windows, a todos los sistemas operativos (principalmente LINUX y UNIX), al tratamiento digital de la imagen y las técnicas de sonido y un largo etcétera. De cara al nuevo año, se incorporarán nuevos cursos, además de los de reciente creación como han sido el curso de Hipertexto, el de Ray Tracing, tecnología WEB y Creación de un Compilador se incorporarán los dedicados a programación de juegos 3D, programación a bajo nivel y Visual Basic 4.0

Creemos que adquirir mensualmente Sólo Programadores es una buena inversión, ya que intentamos estar en la vanguardia de las últimas tendencias en el campo de la programación y cubrir todos los ámbitos de ésta, asistir a todos los eventos importantes de la informática allá donde sean, mantener alguna sección de introducción a la programación dedicada a los más inexpertos que quieran iniciarse en nuestro mundo del código, manteneros informados de las últimas noticias y novedades, y un largo etcétera que pensamos hacen de Sólo Programadores una revista que merece la pena. Sin más, reiteramos nuestras felicitaciones y os recordamos que estamos abiertos a cualquier sugerencia ya sea mediante el correo del lector,teléfono o fax.

Gracias, una vez más, por hacer posible esta revista que muy especialmente este mes va dedicada a todos vosotros.

CORREO DEL LECTOR

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relaccionado con la programación.

P *En mis programas gráficos suelo sincronizarme o esperar al retrazado vertical cuando realizo un scroll o similar para conseguir la mayor suavidad posible y para que la velocidad vaya constante independientemente del proceso realizado. Mi problema es que mientras espero el siguiente retrazado, estoy perdiendo el tiempo y pienso que podría ir rellenando el buffer de la Sound Blaster o cosas así. ¿No existe una interrupción en el hardware de la VGA que me permita sincronizarme con el vídeo? ¿Puedo sincronizarme de alguna otra forma? Muchas gracias y adelante con la revista.*

Ignacio de Burgos Tabernero (Cádiz)

R Algunas tarjetas VGA disponen de una interrupción hardware que se activa a cada retorno del barrido de electrones (retrazado vertical o VBL). La interrupción generada es la 0Ah. Lamentablemente esto no es estándar y no se puede confiar en ella para gestionar un programa, por lo que se debe continuar con el método de la espera mediante la comprobación del bit 3 del puerto 3DAH.
Esto es:

```
while    (port[$3da] and 8) <> 0 do;
while    (port[$3da] and 8) = 0 do;
```

Se espera mientras el bit 3 no esté a 0 (esté a 1) y se continúa esperando mientras se mantenga a 0. Aunque se ha escrito es Pascal, no creo que nadie tenga problemas para comprenderlo.
Para la sincronización o para que un programa se ejecute siempre a la misma velocidad, independientemente de las prestaciones del ordenador, se hace necesario reprogramar el timer del PC para que genere una interrupción cada 1/70 de segundo (la velocidad de refresco de la VGA) y que vaya incrementando un contador. Hecho esto, sólo hay que esperar a que el contador llegue a un determinado valor y con una espera adicional al siguiente barrido, tendremos sincronizado el programa, con el hardware de vídeo y con la velocidad del procesador.

P *En el número 13 de vuestra revista, en la comparativa entre Delphi y Visual Basic, leí que ambas herramientas permiten consultas SQL y que incorporan como motor de bases de datos a ACCESS o bien a DBASE V para Windows. ¿Dispone de algún gestor de bases de datos el compilador Borland C++ 4.0? ¿Puedo acceder a archivos DBF desde los programas compilados por él? En la nueva versión 4.5, ¿es esto posible? Gracias por todo y por último, ¿cómo puedo conseguir algunos números atrasados que me faltan?*

José Carlos Serrano (Córdoba)

R El compilador Borland C++ 4.0 no incluye ningún gestor de base de datos en el paquete, pero se le puede añadir alguna de las librerías especiales para ello que circulan por el mercado. Especialmente para el acceso a los ficheros DBF existen varias, algunas de ellas totalmente freeware. Una de las mejores, y a un precio asequible, es Codebase, que incorpora interfaces para su unión con los compiladores (o ensambladores) más habituales.

La nueva versión 4.5 del compilador Borland sí que incorpora un total control de bases de datos, permitiendo incluso el diseño de aplicaciones con arquitectura Cliente-Servidor.
Para cualquier petición de números atrasados debe remitirse a nuestro departamento de suscripciones, cuyos datos se encuentran en cualquier número de la revista.

P *Antes de nada, quisiera felicitarles por la revista en general y en concreto por esa serie de artículos sobre las demos. Mi pregunta va precisamente sobre ellos. En la mayoría de estos artículos, en el ejecutable que realiza la demo, se suele mezclar código en C o en Pascal con código en ensamblador. Sin embargo hasta ahora no habéis publicado ningún artículo que explique la posible interacción entre módulos programados en distintos lenguajes, por lo que tengo un buen número de preguntas al respecto:*
- ¿Cómo pasa C los parámetros a una función y como los recoge? Necesito saberlo para enlazar rutinas ASM con rutinas C.
- ¿Cómo se accede a las variables globales desde el ensamblador? y ¿cómo las comparte con otros módulos, indicando su disponibilidad o no?
- Cuando C compila un programa, ¿en qué segmento suele colocar las variables globales? y ¿cuál es la estructura general del ejecutable? (a nivel de segmentos)

Adolfo Aladro García (Madrid)

R El lenguaje C dispone de dos sistemas (sólo en algunos compiladores) para el envío de los parámetros a una función:
Envío de los argumentos utilizando la pila.-
Suele ser lo más normal y lo que se genera por defecto, además no tiene ningún limite en el tamaño de los parámetros (sólo el tamaño de la pila). La dirección de retorno de la llamada a la función es el primer elemento de la pila (2, 4 ó 8 bytes, según el modelo de memoria utilizado) y después se guardan los argumentos según el orden de aparición en la función.
Envío de los argumentos utilizando los registros.-
Mucho más rápido que a través de la pila, pero con el tamaño de los parámetros muy limitado. Lo que se suele hacer es almacenar los argumentos posibles en los registros disponibles y el resto en la pila. Los registros utilizados para esta tarea son: AX, BX, CX y DX o su versión extendida (EAX, etc.) en los compiladores de 32 bits.
De todas maneras, lo mejor es declarar las funciones en ensamblador mediante las directivas que definen los procedimientos y los parámetros (PROC, USES, ARGS, etc.) y que el compilador se encargue de acceder correctamente a los argumentos y de colocarlos adecuadamente en la pila.
En ensamblador existen unas directivas para el control de las variables o constantes y su "visibilidad" desde otro módulos, las que interesan a la pregunta son:

PUBLIC [lenguaje] símbolo
Declara a símbolo para manera que sea accesible desde otros módulos. Se puede especificar un lenguaje opcional para que además se encargue de la conversión del nombre, según las normas habituales de "nombrado" de variables en los distintos lenguajes (p.e.- en C se antepone un _ a cada nombre).

EXTERN definición [,definición] ...
definición se compone de
[lenguaje] nombre [contador1]:tipo [:contador2]
Indica que un símbolo está definido en otro módulo. Se especifica su lenguaje origen, su nombre, el tamaño de la cadena (si la hubiera, mediante contador1), el tipo de dato de que se trata y un contador que señala el número de veces que está definido (contador2).

GLOBAL definición [,definición] ...
Es una mezcla de las directivas anteriores, pero no aporta nada nuevo.
La estructura habitual de un programa desarrollado en C es la siguiente:

```
_TEXT segment byte use16 public 'CODE'
assume CS:_TEXT
; aquí metería el código
_TEXT ends
_DATA segment word use16 public 'DATA'
; aquí los datos
_DATA ends
_BSS segment word use16 public 'BSS'
; aquí los datos sin inicializar
_BSS ends
; el clásico agrupamiento
DGROUP group _DATA,_BSS
; estas son las definiciones
de las variables para otros
módulos
extrn _printf
public _main
```

Si es un compilador de 32 bits, entonces declara los segmentos como use32 y si se compila en un modo largo (segmentado), antepone en cada nombre de segmento el nombre del módulo generado: GRAF_TEXT.
Esto no se debe considerar como norma fija, pues siempre depende del compilador y, sobre todo, del sistema operativo para el que se desarrolle. Conviene utilizar el desensamblador de módulos objeto que suele incorporarse en el paquete para ver que genera exactamente el compilador con las distintas opciones.

P *Hola, llevo muchos años en el mundo informático, pero la mayoría del tiempo como usuario (aunque sea avanzado). Desde siempre me he echo la misma pregunta ¿Cómo conseguían "meter" en muchos de los juegos de los ordenadores antiguos (Spectrum, Amstrad, Commodore, etc), doscientas o trescientas pantallas gráficas? Actualmente también se pueden encontrar juegos con una enorme variedad gráfica, la cual es teóricamente imposible albergar en la memoria convencional del PC o en los discos en los que se comercializa el juego. A ver si me pueden quitar la duda.*

Alberto Ocón
(Guadarrama - Madrid)

R El truco no es muy complicado y era totalmente necesario en los ordenadores antiguos que tenían grandes restricciones de memoria. Consiste en el "mapeado" de las pantallas. Estas se componen a base de bloques, y mediante la más o menos afortunada realización del grafista se consigue la perfecta unión de varios de estos bloques para formar un fondo gráfico. Si se dibuja un pequeño bloque de ladrillos, se pueden obtener variadas construcciones con sólo el gasto de un gráfico de 16 x 16 pixels; o si se crea una pequeña tira de césped, otra igual pero con una piedra y otra similar con un arbusto en medio, se puede construir un variado prado en el que sería difícil detectar la repetición gráfica. En algunos de los juegos pioneros se abusada mucho del mapeado de bloques y con sólo 20 pequeños bloques se mapeaban decenas de pantallas, consiguiéndose una variedad gráfica realmente escasa.
En la construcción de los escenarios que utilizan esta técnica se hace imprescindible un programa o utilidad para ir creando las pantallas a base de un juego o conjunto de bloques. Estos programas son auténticos puzzles informáticos y se hace necesaria la presencia de personas dedicadas exclusivamente a mapear.
Esta técnica de obtención de grandes gráficos a partir de sub-bloques es utilizada hasta en los sprites o figuras animadas en las que el movimiento de un personaje caminando se consigue con un par de gráficos del torso, una cabeza y tres o cuatro secuencias de las piernas.

YA A LA VENTA EL NÚMERO 2 DE LA PRIMERA REVISTA INTERNACIONAL PARA USUARIOS DE INTERNET



ABETO EDITORIAL

Coja el ratón. Recorra la pantalla.

Entre en una base de datos. Deténgase.

Avance con una transacción.

Haga una jugada redonda en multimedia. Arrastre. Suelte. "Clic" y...

...¡Gooooooooool!

Es fácil ver como nada le puede lanzar tan de lleno a la programación orientada a objetos como VisualAge C++ V. 3.0, de IBM.

VisualAge C++ forma parte de la familia de Herramientas de Alta Productividad junto con VisualGen, VisualAge Smalltalk, VRPG, Visualizer...

Va más allá de la simple programación visual.

Este entorno de desarrollo integrado le permite generar aplicaciones Cliente/Servidor rápidas y ajustadas. Y tan fácil como pulsar un botón.

Un simple "arrastrar y soltar" incorpora todas las posibilidades que la librería de objetos predefinidos Open Class aporta. Esto significa que puede crear aplicaciones Cliente/Servidor distribuidas rápidamente.

¿Puede su software llegar a tanto?

Con Open Class y compiladores C++ para OS/2, Sun Solaris, OS/400, AIX y MVS (y versión Windows en preparación), utilizar sus nuevas aplicaciones orientadas a objetos a través de múltiples plataformas es muy fácil.

Creemos que estas nuevas funciones son realmente alucinantes.

La revista InfoWorld así lo piensa y define al nuevo VisualAge C++ como "una obra de arte de la programación visual" y "lo mejor en reutilización de objetos".

El desarrollo de aplicaciones Cliente/Servidor orientado a objetos con VisualAge C++ para OS/2, puede ser su mejor jugada.



Si desea más información, acuda a su Concesionario Autorizado más próximo o llámenos al 901 100 400 (9:00 a 19:00 h. de Lunes a Viernes).

IBM Internet: http//www.software.ibm.com

Adquiera su VisualAge C++ para OS/2 y benefíciese de un kit de desarrollo que incluye OS/2 Warp y un CD ROM con una copia de evaluación de DB2 V.2.

IBM

Soluciones para nuestro pequeño mundo

Esta es una oferta limitada en el tiempo. Para recibir su kit de desarrollo simplemente complete y devuelva la carta de registro a la dirección que aparece en la misma antes del 12 de Enero de 1996. OS/2 Warp será enviado únicamente en CD Rom. VisualAge C++ para OS/400 estará disponible en el cuarto trimestre de 1995.

IBM, OS/2, DB2, OS/400, VisualAge, Visualgen, VRPG y Visualizer son marcas registradas de International Business Machines Corporation. Se reconocen todas las otras marcas registradas. ©1995 IBM.